Requirements Eng (2005) 10: 34–44
DOI 10.1007/s00766-004-0194-4

ORIGINAL ARTICLE

Guttorm Sindre · Andreas L. Opdahl

Eliciting security requirements with misuse cases

Received: 15 February 2002 / Accepted: 5 March 2004 / Published online: 24 June 2004
© Springer-Verlag London Limited 2004

Abstract Use cases have become increasingly common during requirements engineering, but they offer limited support for eliciting security threats and requirements. At the same time, the importance of security is growing with the rise of phenomena such as e-commerce and nomadic and geographically distributed work. This paper presents a systematic approach to eliciting security requirements based on use cases, with emphasis on description and method guidelines. The approach extends traditional use cases to also cover misuse, and is potentially useful for several other types of extra-functional requirements beyond security.

Keywords Security requirements · Use cases · Scenarios · Extra-functional requirements · Requirements elicitation · Requirements determination · Requirements specification · Requirements analysis

1 Introduction

Use cases [1–3] have become popular for determining, communicating, specifying, and documenting requirements [4, 5]. Many groups of stakeholders turn out to be more comfortable with descriptions of operational action sequences than with declarative specifications of software requirements [5]. An industrial survey [6] reports that scenarios are useful for determining and validating requirements, and for making them concrete, agreed-on, and consistent.

G. Sindre (✉)
Department of Computer and Information Science,
Norwegian University of Science and Technology (NTNU),
Trondheim, Norway
E-mail: guttors@idi.ntnu.no

A. L. Opdahl
Department of Information Science and Media Studies,
University of Bergen, Norway
E-mail: Andreas.Opdahl@uib.no

But there are also problems with use-case-based approaches to requirements engineering, such as oversimplified assumptions about the problem domain [7] and premature design decisions [2, 8]. Use cases are suitable for most functional requirements, but may lead to neglect of extra-functional requirements, such as security requirements. Alarmingly, such practices have also been observed in projects developing software systems with substantial security needs, such as e-commerce software [9]. Compounding the problems with use-case-based approaches, security requirements may be poorly treated because traditional methods and standards for security engineering [10, 11] are heavyweight and hard to understand. It has also been observed that industrial security approaches derive from the solution world rather than the problem world [12], whereas good requirements engineering practice mandates a thorough understanding of the problem before suggesting solutions. Hence, both use-case-based and other approaches to requirements engineering would benefit from a closer integration between informal and formal approaches, and between functional and extra-functional requirements work [13–16].

It turns out that, with slight modifications, use cases can aid the integration of functional and extra-functional requirements work when considering security requirements: in our previous work, we have extended *positive* (regular) use case diagrams with *negative* use cases—*misuse cases*—that specify behavior *not* wanted in the proposed system for the purpose of eliciting security requirements [17, 18]. After all, even an unwanted interaction sequence is still an interaction sequence, and many security breaches can be described in a stepwise fashion that resembles ordinary use cases [17–24]. The major difference is that a use case achieves something of value for the system owner and its stakeholders, whereas the security breach is harmful.

In this paper, we first explain the *basic concepts and notation* for misuse cases (Sect. 2). We then present guidelines for how to *describe misuse cases in detail* using

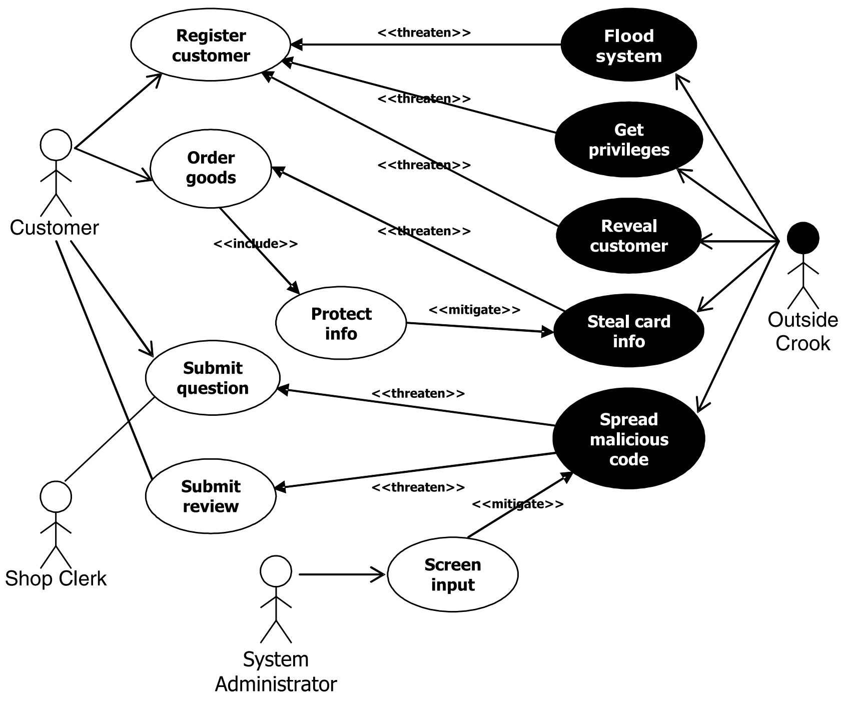

Fig. 1 Example use and misuse cases for an e-store

textual templates (Sect. 3) and *method guidelines* for eliciting security requirements with misuse cases (Sect. 4). Next, we review the *practical use* of misuse case analysis (Sect. 5), discuss the *strengths and weaknesses* of misuse cases (Sect. 6), and compare them to *related work* [19–22, 25, 26] (Sect. 7). Finally, we conclude the paper and suggest paths for further work (Sect. 8) [17, 22]. The main contribution of the paper is to present, in a coherent manner, all of the past individual contributions and to emphasize description and method guidelines in particular.

2 Concepts and notation

In line with the UML definitions of *use case* and *actor* [27], we define *misuse cases* and *misusers* as follows[1]:

Misuse case A sequence of actions, including variants, that a system or other entity can perform, interacting with misusers of the entity and causing harm to some stakeholder if the sequence is allowed to complete[2].

Misuser An actor that initiates misuse cases, either intentionally or inadvertently.

Figure 1 uses inverted graphics to show misuse cases together with regular use cases in a high-level specification of part of an e-shop software system. Compared to regular use cases, the inverted notation indicates both similarity (because the same symbol shapes are used) and negation (because of the inverted graphics). Use and misuse cases can, thereby, be shown in the same diagram without confusion.

Ordinary use case relationships such as *include*, *extend*, and *generalize* can be used between misuse cases too, and ordinary *association* relationships can be used between misusers and their misuse cases. There are also more specific relationships between use and misuse cases:

Use case mitigate misuse case The use case is a countermeasure against a misuse case, i.e., the use case reduces the misuse case's chance of succeeding. An example is "protect info", which mitigates "steal credit card info", as shown in Fig. 1.

Misuse case threaten use case The use case is exploited or hindered by a misuse case. For example, the "register customer" use case is threatened by a denial-of-service attack, "flood system", that prevents legitimate users from accessing internet services, including customer registration.

Figure 2 shows a metamodel of the basic misuse-case concepts and their relation to the UML metamodel [27]. In addition to **Actors** and **Use cases**, the model introduces **Misusers** and **Misuse Cases**, so that misuse cases may **Threaten** regular use cases. The model also identifies **"Security Use Cases"**, which may **Mitigate** misuse cases. Hence, whereas regular use cases represent requirements in general, misuse cases represent *security*

[1]Although the definitions and the metamodel in this section are aligned with the UML, misuse cases do not inherently depend on the UML and can augment other use case techniques equally well.
[2]For simplicity, we will use the term *action sequence* in this paper, although we often talk about sequences of both *actions* and *interactions*. Some authors prefer the term "step" where we use *action*.

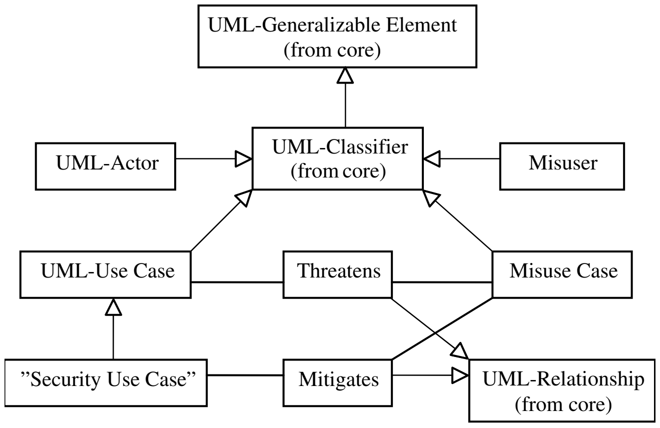
UML-Generalizable Element (from core)
UML-Actor
UML-Classifier (from core)
Misuser
UML-Use Case
Threatens
Misuse Case
"Security Use Case"
Mitigates
UML-Relationship (from core)

Fig. 2 A metamodel of the basic concepts presented in this section and their relation to the UML metamodel [28]. Existing constructs in the metamodel are prefixed with *UML-*

threats and "security use cases" represent *security requirements*, i.e., countermeasures that mitigate the threats.

The metamodel does not propose **"Security Use Cases"** as a new abstract metaclass in the UML metamodel at this time, hence the quotation marks. Also, although we could have introduced a new abstract metaclass, **"Use Or Misuse Case"**, to account for attributes that are shared between use and misuse cases, we leave this to be decided in further work[3]. In further work, the ***Threaten*** relationship could also be defined as a specialization of the UML's regular **Extend** relationship[4]. Finally, the metamodel does not explicitly show UML **Generalizations**, which can be used between two **GeneralizableElements**, or **Associations**, which can be used between two or more ***Classifiers***.

3 Textual specification of misuse cases

A use-case diagram only gives an overview of the required system functionality, so the essence of a use case is usually captured in the associated textual description [3]. Textual descriptions also play an important part when representing misuse cases. This section presents templates for describing misuse cases textually [18, 28] and discusses how to use the templates to elicit security requirements. Templates are important because they encourage developers to write clear and simple action sequences. They are also interesting for researchers because they offer support for controlled and repeatable empirical evaluation of description and method guidelines. We identify two ways of expressing misuse cases textually; a *lightweight* description that is embedded in the textual description of a use case, and an *extensive* description of misuse cases on equal terms with ordinary use cases.

3.1 Lightweight misuse case description

The lightweight approach embeds the description of misuse within a regular use-case template, such as the templates proposed by Kulak and Guiney [5], Cockburn [3], or RUP [29], by extending them with a field called **Threats**. In Table 1, the **Threats** field of the "register customer" use case contains a threat, **T1**, which twists the customer's form submission (action 3) so that the information entered does not describe the person actually entering it. The last of the three possible outcomes of **T1** corresponds to the "reveal customer" misuse case of Fig. 1.

For those who prefer writing use cases with separate columns for the user and system interactions, threats can conveniently be represented as an additional, third column instead of an additional field. Table 2 extends Constantine and Lockwood's [2] essential use case, *gettingCash*, with a **Threats** column. This representation makes it even clearer against which action in the sequence the threat is directed.

For systems where security is important, the regular use-case template should be extended with either a **Threats** field or a column: a filled-in threats field/column makes the information easy to detect, whereas an empty threats field/column indicates that security issues still need to be investigated. When all of the possible threats to a use case have been explicitly considered and found unimportant, this should, therefore, be expressed explicitly in the **Threats** field, e.g., "probability of misuse considered extremely low" or "impact of potential misuse assumed harmless".

3.2 Extensive misuse case description

To support detailed determination and analysis of security threats, we propose to describe misuse cases extensively—on equal terms with regular use cases—based on a template of fields that are mostly described using text, such as triggers, preconditions, basic, and alternative paths. Most of the fields in the use-case templates of Kulak and Guiney [5], Cockburn [3], and RUP [29] are also relevant for describing misuse cases, but some adaptations are necessary and some new fields must be introduced. For a detailed discussion, see [18].

In an extensive misuse-case description, the fields **Name**, **Summary**, **Author**, and **Date** retain the same meaning as in regular use cases. The **Basic** and **Alternative path** fields for misuse cases now describe the sequences of actions that the misuser(s) and the proposed system go through to cause harm. The other fields in extensive misuse-case descriptions are explained in

[3]A corresponding abstract metaclass *Actor Or Misuser* is also possible.
[4]"Mitigate" replaces the "prevent" and "detect" relationships, and "threaten" replaces the "extend" and "include" relationships defined in [17], following a suggestion made by Alexander [22] in both cases.

Table 1 A lightweight misuse case description embedded in the use case, "register customer", from Fig. 1. (Of course, more threats can be described in addition to **T1**)

Name:	Register customer
Iteration:	Filled
Summary:	The customer registers for the e-shop, giving name, address, email, and phone
Basic path:	bp–1. The customer selects to register bp–2. The system provides the registration form bp–3. The customer completes the form and submits bp–4. The system acknowledges registration, returning a customer reference number
Alternative paths:	[...]
Exception paths:	**E1.** In action 3, the customer submits with mandatory information missing. Return to action 3 to provide more info **E2.** In action 3, the submitted info matches an already registered customer. The system notifies the user that registration is abandoned because the customer is already registered. This ends the use cases
Extension points:	[...]
Triggers:	[...]
Assumptions:	[...]
Preconditions:	[...]
Postconditions:	The customer is now registered, and will be enabled to order goods from the e-shop without providing contact info anew
Related business rules:	[...]
Threats:	**T1:** The customer is not registering with his own name and address, but with an assumed identity. Possible outcomes: T1–1. A non-existing person is registered as customer T1–2. An existing person is unwillingly and unknowingly registered as a customer T1–3. It is revealed to a third party that the named person is a customer of the e-shop (see exception path **E2** above)**T2:** [...]
Author:	John Davis
Date:	2001.05.23

Table 2 A lightweight misuse-case description embedded in the regular use case, *gettingCash* from an ATM

gettingCash		
User intention	System response	Threats
Identify self		Identity spoofedIdentification spied on
	Verify identity	ATM tampered with
	Offer choices	
Choose		
	Dispense cash	
Take cash		Customer is robbed

Table 3. The next section presents guidelines for describing misuse cases in detail, pointing out that the full set of fields is only advocated for misuse cases that describe security-critical parts of the proposed system in fine detail.

Table 4 shows an extensive description of the misuse case, "Tamper with database by web query manipulation", which specializes the general "Tamper with data" from Fig. 1. "Tamper with data" cannot be described as a single coherent action sequence because it can be achieved in several different ways.

4 Working with misuse cases

Misuse-case diagrams and the associated textual templates inform developers only about which security-related information they should specify and not about how and when to do so. Also, misuse-case diagrams and templates say nothing about how the security requirements process is related to other software development activities, nor about when to use lightweight and when to use extensive misuse-case specifications. This section, therefore, provides guidelines for working with misuse cases. In addition to being useful for developers, the method guidelines are important for research on misuse cases because they are starting points for evaluating misuse cases empirically.

4.1 The security requirements process

We propose the following five steps for eliciting security requirements with misuse cases [30]:

1. *Identify critical assets* in the system, where an asset is either information that the enterprise possesses, virtual locations that the enterprise controls, or computerized activities that the enterprise performs [30].
2. *Define security goals* for each asset, preferably aided by a standard typology of security goals, such as the one inherent in the common criteria for IT security evaluation [10][5]

[5]The common criteria is intended as a *security evaluation standard*, but because it is organized according to classes, families, and components of security criteria, and because security criteria can be seen as operational security goals, the common criteria also entails a *typology of security goals*.

Table 3 The full list of text fields for describing misuse cases extensively. Most extensive misuse case descriptions will only use a subset of fields from this list

Name, **Summary**, **Author**, and **Date**:	These fields retain the same meaning as in regular use cases
Basic path:	This field describes the actions that the misuser(s) and the system go through to harm the proposed system
Alternative paths:	This field describes ways to harm the proposed system that are not accounted for by the basic path, but are still sufficiently similar to be described as variants of the basic path
Mitigation points:	This field identifies those actions in a basic or alternative path where misuse can be mitigated. Several ways to mitigate misuse of a particular action can be described in the same field and each of them may be further described in a separate security use case. As for extension points, the misuse case must eventually have a mitigate relationship to a corresponding security use case. However, the detailed description of security use cases is optional, because it is often closer to design, requiring detailed analysis of risks and implementation costs that go beyond use and misuse cases
Extension points:	In some cases, a misuse case may be extended with optional paths whose details are described in a separate extension misuse case. This field lists the actions in the main or alternative paths where optional paths may be inserted. As for extension points in regular use cases, the misuse case must have an extend relationship to the misuse case that contains the optional path
Trigger:	This field describes the states or events in the system or its environment that may initiate the misuse case. For some misuse cases, the trigger is just the predicate True, indicating a permanently present danger
Assumptions:	This field describes the states in the system's environment that make the misuse case possible
Preconditions:	This field describes the system states that make the misuse case possible
Mitigation guarantee:	This field describes the guaranteed outcome of mitigating a misuse case. If the mitigation points are not yet specified in detail, the mitigation guarantee describes the level of security required from the mitigating security use cases that will be designed later. When the mitigation points in the misuse case have been detailed by security use cases, this field describes the strongest possible security guarantee that can be made, regardless of how the misuse case is mitigated
Related business rules:	Typically, business rules will be violated by the misuse. This field contains links to such rules, maybe along with links to rules that enable the threat or that limit how it could be mitigated or eliminated
Misuser profile:	This field describes whatever can be assumed about the misuser, for example, whether the misuser acts intentionally or inadvertently; whether the misuser is an insider or outsider; and how technically skilled the misuser must be
Scope:	This field indicates whether the proposed system in a misuse case is, e.g., an entire business, a system of both users and computers, or just a software system
Iteration:	As for regular use cases, it is useful to allow both initial and detailed descriptions of misuse cases. This field indicates the misuse case's iteration level, usually taken from the set of iteration levels used for the use cases in the project
Level:	As for regular use cases, misuse cases can be specified at a general or specific abstraction level. This field indicates whether the misuse case is, e.g., a summary, a user goal, or a sub-function, following [3]
Stakeholders and risks:	This field specifies the major risks for each stakeholder involved in the misuse case. On an abstract level, risks can be described textually, e.g., "the system is unavailable for several hours" or "a competitor gets hold of sensitive medical data about an applicant". On a concrete level, the likelihood and cost of each misuse variant can be estimated, where the cost includes potential losses, should the threats come true
Technology and data variations:	A misuser may carry out a misuse case from a variety of technical platforms, such as a PC or a WAP phone and, since only a few equipment-related actions will differ in each case, it is unnecessary to specify two separate paths. Instead, this field lists the candidate types of equipment and explains how they differ in particular actions
Terminology and explanations:	This field contains explanations of technical terms and other issues

3. *Identify threats* to each security goal by identifying stakeholders that may intentionally harm the system or its environment and/or identifying sequences of actions that may result in intentional harm[6]
4. *Identify and analyze risks* for the threats using standard techniques for risk analysis and costing from the security and safety engineering fields [31–33]
5. *Define security requirements* for the threats to match risks and protection costs, preferably aided by a taxonomy of security requirements, such as [10, 11]. This process of identifying critical assets, threats, and security requirements (or countermeasures) is *cyclical*. On the one hand, critical assets defined in the larger process drive the identification of threats in the security process. On the other hand, the threats identified in the security process drive the definition

[6]Most of the techniques and methods proposed in this paper may apply equally well to *unintentional* harm, but that would lead us into the area of *safety requirements*. The definition of misuse cases given in Sect. 2 explicitly allows for both intentional and inadvertent—or unintentional—misuse.

Table 4 The misuse case, "Tamper with database by web query manipulation"

Misuse case name:	Tamper with database by web query manipulation
Summary:	A crook manipulates the web query, submitted from a search form, to update or delete information, or to reveal confidential information
Author:	David Jones
Date:	2001.02.23
Basic path:	bp–1. The crook provides some values on a product search form and submitsbp–2. The system displays the product(s) matching the querybp–3. The crook alters the submitted URL, introducing a query error, and resubmitsbp–4. The query fails and the system displays the database error message to the crook, revealing more about the database structurebp–5. The crook alters the query further, for instance adding a nested query to reveal secret data or update or delete data, and submitsbp–6. The system executes the altered query, changing the database or revealing content that should have been secret
Alternative paths:	**ap1.** In action 3 or 5, the crook does not alter the URL in the address window, but introduces errors or nested queries directly into form input fields
Mitigation points:	**mp1.** In action 4, the exact database error message is not revealed to the client. This will not entirely prevent the misuse, but the crook will have a harder time guessing table and field names in action 5**mp2.** In action 6, the system does not execute the altered query because all queries submitted from forms are explicitly checked in accordance with what should be expected from that form. This prevents the misuse case
Extension points:	[...]
Triggers:	**tr1.** Always true. This can happen at any time
Preconditions:	**pc1.** The crook is able to search for products, either because this function is publicly available, or by having registered as a customer
Assumptions:	**as1.** The system has search forms feeding input into database queries
Mitigation guarantee:	The crook is unable to access the database in an unauthorized manner through a publicly available web form (see mp2)
Related business rules:	The services of the e-shop shall be available to customers over the internet
Potential misuser profile:	Skilled. Knowledge of databases and query language, or at least able to understand published exploits on cracker web sites
Stakeholders and threats:	**st1.** e-shop: loss of data if deleted. Potential loss of revenue if customers are unable to *Order Product*, or if prices have been altered. Bad will resulting from customer problems in st2**st2.** customers: potentially losing money (at least temporarily) if crook has increased product prices. Unable to order if data lacking, wasting time. Also, more far-reaching issues of loss of privacy (if misuser reveals confidential information about customers) or even money loss (if misuser reveals, e.g., credit card numbers)
Terminology and explanations:	[...]
Scope:	Entire business and business environment
Abstraction level:	Misuser subgoal
Precision level:	Focused

of new security requirements which, when implemented, may create new vulnerable assets (of the *computerized activity* type). This cycle has been investigated by Alexander [22].

The security requirements process can be supported by a *repository* of reusable security threats and associated security requirements represented, respectively, as misuse cases and security use cases, or in other ways [30].

4.2 Misuse cases in the larger development process

The five-step security requirements process does not stand alone, but must be embedded in a software development process, which provides a context for defining security goals and identifying and analyzing risk. Misuse cases feature most prominently in three of the five steps:

– In step 3, the security threats identified can be described as misuse cases and misusers, although, in general, the above steps are independent of particular ways of representing security threats and requirements [30]. The best way to specify security threats depends on the situation, as is discussed later in this section.
– In step 4, the relationships identified between misuse cases can aid risk analysis. For example, [28] points out that extend/include relationships and generalization relationships, respectively, are analogous to AND and OR nodes in fault trees and similar trees.
– In step 5, the security requirements defined are specified either as independent security use cases or in the mitigation fields of extensively described misuse cases [30]. Again, the best way to specify security requirements depends on the situation and is discussed later. There are many ways to determine and specify requirements with use cases, and it is not the purpose of this paper to tie misuse cases tightly to one of them. On the contrary, given the contingent and opportunistic nature of early requirements determination [34], overly detailed prescriptive method guidelines are inappropriate for both use and misuse cases. Instead, developers must be ready to use the available techniques differently, depending on the situation.

4.3 Specifying misuse cases in detail

Guidelines are also needed for using the textual templates presented in Sect. 3, in particular, regarding the choice between *lightweight* and *extensive* descriptions. Although the lightweight approach offers a quick, simple, and systematic way of eliciting security threats, it does not aid developers in analyzing those threats in detail. As a consequence, it becomes difficult to find appropriate bundles of security requirements—possibly expressed as security use cases—that best match each threat.

Lightweight descriptions are, therefore, better early in development, when brainstorming to get an overview of the threats faced by the system. Lightweight descriptions are also more appropriate for misuse cases believed to be less critical for overall security and for misuse cases that are slight "twists" on a regular use case. Such twists can be described in much the same way as exceptional use-case paths, as long as they remain uniquely identified, searchable, and traceable elements in the specification. Extensive descriptions are better for later development stages and when specifying that a particular misuse case is mitigated by certain corresponding requirements use cases. In many cases, threats that are first specified lightly will later be described extensively. Finally, the choice between lightweight and extensive description depends on how complex the misuse is. Simple misuse involving just a single malicious action can be described in lightweight format, whereas misuse involving intricate action sequences and alternative paths calls for extensive description.

Many of the other concerns when specifying misuse cases in detail are similar to those for regular use cases. As for use cases, developers should describe each misuse case independently of particular technological solutions whenever this is possible. For example, misuse-case actions should not mention particular security mechanisms like passwords, firewalls, and encryption [26]. Also, as for use cases, developers should first describe each misuse case in little detail and then gradually make the description more detailed, concentrating effort on the higher-risk misuse cases. It is hard to provide more precise guidelines on granularity because some misuse cases will threaten the entire software system, and others just single use cases or single use-case actions.

5 Validation

The misuse case notation has already been validated in several research and industrial projects:

- The authors have used misuse cases to elicit and initiate discussion about functional, security, and safety requirements for a knowledge map application in an EU-funded research project[7]. The approach has, so far, turned out to be easy-to-understand and useful for eliciting requirements both for the planned software and for organizational use guidelines.
- Another EU-project[8] embedded misuse cases in an integrated model-based framework for risk management to investigate whether a design is secure against identified threats. The framework was used in six projects in the e-shop and telemedicine domains [35, 36].
- Breivik [37] has used misuse cases to represent security threats ("attack components") from the Open Web Application Security Project (OWASP) [38] in pattern form. The patterns were validated in interviews with a variety of stakeholders, indicating that the notation was easy to understand and might be useful to facilitate communication and understanding about security in the early development stages. Breivik's [37] investigation has raised a host of other issues for further research.
- Alexander [22] has used misuse cases to analyze requirements trade-offs. He reports that misuse case diagrams contributed to successful determination of threats and requirements, and to subsequent resolution of design conflicts. The notation was easy to understand. There is still a need for more conclusive validation of misuse cases in large-scale industrial settings.

6 Discussion

Although misuse cases is an interesting new approach, it is only one of many ways to elicit security requirements, is an approach that is not always appropriate, that must be adapted to the situation at hand, and that must often be used in combination with other techniques. In order to know when to use misuse cases, how to adapt them to the situation, and with which other techniques to combine them, it is necessary to understand their strengths and weaknesses.

6.1 Strengths

As indicated by the validations, misuse cases allow *early focus on security* by describing security threats and then requirements, without going into design. In particular, the informal nature of misuse cases encourages *analyst and stakeholder creativity* and promotes *user/customer assurance and education* by omitting technical security details, thereby letting stakeholders at different levels of technical competence discuss threats in a way that they can all understand. Looking at the system from a

[7]Project 508011 INTEROP (Interoperability Research for networked enterprises, applications, and software) is a network of excellence under the sixth framework program (IST). It has 50 partners from various European countries.

[8]The CORAS project (IST-2000-25031) addressed risk assessment of security critical systems. It had 11 partners from four countries (UK, Germany, Greece, and Norway). The technical coordinator was SINTEF (N) and the administrative coordinator was Telenor (N). The project was successfully completed in September 2003.

misuser perspective further increases the chance of *discovering threats that would otherwise have been ignored.* Moreover, while formal methods certainly have a place in safety and security engineering, the expert interview study reported by Hickey and Davis [39] did not recommend formal methods for *elicitation*: even for safety-critical systems, formal methods were considered to distance stakeholders too much from the elicitation process. Coughlan and Macredie [40] also stress the importance of using representations that may be understood by the stakeholders, because *effective communication* is crucial to the successful outcome of the requirements process.

The visualization of links between use cases and misuse cases will help *organize the requirements specification* so that related functional and extra-functional requirements are linked to one another [15]. When functional requirements are specified with use cases, the challenge is to link each use case to its related extra-functional requirements [5]. Misuse cases solve this problem for security (and perhaps also, safety) requirements because functional and extra-functional requirements are represented in similar ways and because misuse-case diagrams link regular use cases to both threats and potential countermeasures. This will also aid the *prioritization of requirements* because the real cost of implementing a use case includes the protection needed to mitigate all serious threats to it. If security is dealt with later, or documented separately from use cases, prioritization might not properly consider the induced security costs.

Also, the links will support the *tracing of security requirements* to the threats that motivated them. In addition to explaining requirements and design choices, traces are important for proper *change management* [15], which is particularly important for continuous *security management* [32] when threat situations change quickly and unpredictably as new software vulnerabilities are published and new cracker software is distributed on the web.

When use and misuse cases are represented at a generic level, they can be easily *reused* to give new development projects a flying start in identifying security threats and corresponding security requirements. Reusing misuse cases and security requirements is discussed further in [30].

6.2 Weaknesses

Misuse cases and misuse-case-supported security requirements analysis also suffer from a number of weaknesses. Most importantly, the *open-ended method guidelines* mean that developers will have to improvise. This is a potential problem in security engineering, where formal methods are recommended [33]. Until more detailed and formal guidelines are offered, security defects may be introduced by the security requirements process itself, by the integration of the security process within the embedding software development process, and by the detailed textual descriptions of use and misuse cases.

A particular problem introduced by weak method guidelines is that the potentially large number of threats that must be considered may lead to *analysis paralysis.* This weakness is more of a problem with the security requirements process in Sect. 4 than with the concept of, and notation for, misuse cases. Reuse of security threats and requirements may alleviate the problem somewhat, but the guidelines for how to prioritize and when to stop the security analysis must also be developed further.

In addition, misuse cases are *not equally suitable for all kinds of threats*, focusing mainly on misuse where an identifiable attacker performs a harmful sequence of actions by exploiting another sequence of actions supported by the system:

- Firstly, the misuse is not always an identifiable sequence of actions, like when misuse is achieved in a single operation such as e-mailing a virus. Although this kind of misuse can still be described as a misuse case with many template fields filled in, the path fields will remain empty, making the detailed description less informative and automated pattern retrieval more difficult.
- Secondly, there is not always an identifiable misuser, like when a virus spreads from computer to computer independently of its creator, who is no longer in control of the process. Although there are no clearly identifiable misusers in these situations, there are possible solutions: either the virus itself is considered the misuser—a solution that could be used for other kinds of software too, e.g., attack script engines and automated agents—or the naïve user who inadvertently runs the virus is considered the misuser—a solution that would extend to inadvertent misuse in general and, thereby, to safety requirements.
- Thirdly, the misuse does not always exploit an identifiable sequence of actions and, although the misuse case and the misuser may be identifiable, it is not possible to identify a regular use case that is threatened by the misuse. A virus attack is again a case in point, because the virus may, in principle, enter the system through any kind of data transmission into the system and may, afterwards, affect any action taken by the system. Finally, the fairly limited experience with practical applications of the misuse case technique is itself a weakness that must be addressed by further work.

7 Related work

Several other authors have suggested or discussed negative use cases or scenarios in relation to security. Ellison et al. [25] introduce "*intruders*" and "*intrusion scenarios*" in their case study of part of a large-scale distributed healthcare system. Their study focuses on survivability requirements analysis, an area that subsumes security, safety, and reliability. Intruders and intrusion scenarios are similar to misusers and misuse cases, respectively, but Ellison et al. [25] do not provide a diagram notation, a description template, or general method guidelines for intrusion scenarios.

McDermott and Fox [19, 20] propose "*abuse cases*", which are similar to our proposal. Abuse cases are complementary to misuse cases, because McDermott and Fox [19] focus specifically on security requirements and their relation to design[9] and testing, whereas our approach focuses on elicitation of security requirements in relation to other requirements. McDermott and Fox [19] do not show "use" and "abuse cases" in the same diagram, so no relationships between use and abuse can be depicted either. They also define an abuse case as a *family* of cases that can be achieved in different ways, whereas our approach would use *generalization*[10].

Potts [21] distinguishes between "*abuse cases*" that violate policies and "*misuse cases*" that willfully undermine a policy, e.g., using information for another purpose than it was gathered for. Potts thereby suggests a more fine-grained terminology than ours. In the context of goal-oriented requirements engineering, Potts [41] also introduces "*obstacles*", i.e., exceptional conditions that prevent goal fulfillment. In relation to our work, security goals can be seen as a kind of general goal and security threats as a kind of obstacle. Obstacles are investigated more closely in relation to security policies by Anton and Earp [42], and they are elaborated and formalized by Lamsweerde and Letier [43], who propose heuristics for obstacle identification and strategies for adding new goals to resolve obstacles. In relation to our work, adding new goals to resolve obstacles is similar to adding security use cases to mitigate misuse cases. However, Lamsweerde and Letier [43] do not propose a diagram notation or a template for the textual representation of obstacles, which are, instead, represented as formal logic assertions—an approach that complements our focus on actors and the action sequences they perform to achieve their goals.

Alexander [22] has used misuse cases as presented in this paper in a practical setting, providing useful experience and suggesting paths for further work. Although Alexander used our diagram notation, his way of working with misuse cases was different. Whereas [17, 18] focused on misuse cases, how they threaten regular use cases, and how they are mitigated by security use cases, Alexander also considers how the security use cases can, in turn, be threatened by new misuse cases. On the other hand, Alexander does not consider a structured description of misuse cases in detail.

[9]The design angle is particularly evident in [20], where assurance arguments are used to show that the design really satisfies the requirements.

[10]Finally, there are differences in the textual templates: our template describes misuse case actions, exceptions, mitigations, etc. in more detail, whereas McDermott and Fox' [19] templates provide more detail about the attacker.

Firesmith [26] proposes a template for "*security use cases*", which are different from misuse cases because they represent security-related requirements rather than threats. Firesmith's [26] security use cases have been adopted by us, although we have not yet included his template. Sindre et al. [30] proposes an approach to reusing security requirements that uses security use cases and misuse cases together.

8 Conclusion and further work

Use cases are popular tools for eliciting functional requirements but less suited for extra-functional requirements—such as security requirements—that describe behaviors *not* wanted in the system. This paper has proposed two new concepts—*misuse cases* and *misusers*—along with suitable relationships, a diagram notation, templates for textual descriptions, and method guidelines. The approach has been tested on examples and in realistic settings [22, 35, 36], and it is currently used in research on *risk management* and on *security patterns*.

Compared to other similar approaches, misuse cases integrate more closely with regular use cases and, thereby, facilitate better analysis of how functional requirements relate to security threats and requirements. The proposed template is also more comprehensive than comparable approaches. Method guidelines are provided to ensure that the approach is helpful in the early elicitation of security requirements. Important strengths of the proposed approach include ensuring early focus on security, encouraging analyst and stakeholder creativity, promoting user/customer assurance and education, supporting explicit prioritization, better organization and better tracing of requirements, facilitating proper change management, and providing support for reusing misuse cases and associated security requirements. However, the method guidelines provided are still too general and imprecise; the number of potentially critical assets and associated threats that must be considered is, therefore, large, and the misuse-case approach itself is not equally suitable for all kinds of threats, specifically because misuse does not always involve or exploit an identifiable sequence of actions nor an identifiable misuser. Whereas some of these weaknesses reflect inherent trade-offs that must be judged according to the situation at hand, other weaknesses mainly call for more work—in particular on providing more detailed method guidelines.

An obvious candidate for further work is to evaluate misuse cases further in industrial settings. The approach is well suited for industrial evaluation because it extends standard OO concepts and it is linked to the UML. It should be easy to incorporate misuse cases in a use-case-based software development organization, because the proposed extensions are small and simple to implement, and because the proposed template resembles regular use-case templates.

Another important goal for further work is to facilitate broader industrial adoption of misuse cases. For this to happen, misuse-case analysis must be embedded in well-documented and tool-supported RE methods that are *use-case-driven* (because misuse cases integrate well with regular use cases), e.g., [44–46], *goal-oriented* (because security threats can be considered anti-goals or obstacles), and/or *lightweight* (because many software development organizations already employ, or are considering employing, agile methods [47] and the misuse case notation induces little overhead). In particular, the goal-oriented i* approach has recently been used for representing trust, attacks, and countermeasures [48]. Industrial adoption of misuse cases is also likely to be based on *reuse* of security threats and requirements, and work is in progress on establishing and evaluating a library of *security misuse case patterns* and on using them to determine security requirements in practical settings.

Another interesting direction is to align misuse cases with other security approaches. Our approach can be integrated with existing security standards [10, 11], which classify threats as separate from other security issues, such as objectives and requirements. For example, the 29 threat categories identified in [11] could potentially aid in discovering a more complete determination of security requirements. Because existing standards are often written in a formal and technical style, integrating them with our diagram notation and method may make the standards more readily available to end-users and developers.

Another possibility is to align misuse cases with traditional fault-tree analysis methods from the safety area, such as threat trees [49] or attack trees [50, 51]. Adapted for security analysis, these trees would decompose security threats using AND and OR nodes, where OR nodes correspond directly to misuse case *generalization*—such as when a password can be obtained in several ways [28]—and where AND nodes are not explicitly covered in our notation (but may be implicit in misuse-case *inclusion* and *preconditions*). Extending the misuse case notation with AND nodes, e.g., using the UML's aggregation symbol, may be straightforward, and a natural next step would be to align the misuse cases with the NFR framework [52] for analyzing non-functional requirements.

A further interesting direction is to consider misuse cases in relation to other types of extra-functional requirements such as safety, privacy, and usability, all dealing with behavior *not* wanted in the proposed system. For example, better integration of informal and formal techniques is necessary for progress in safety analysis [53]. Misuse cases should also be complemented with techniques for risk analysis and costing from security and safety engineering [31–33].

In the meantime, misuse cases can be used as an informal and integrating front-end to more heavyweight techniques, making it easier for various stakeholders to participate in eliciting security requirements for new information and software systems.

References

1. Jacobson I et al (1992) Object-oriented software engineering: a use case driven approach. Addison-Wesley, Boston
2. Constantine LL, Lockwood LAD (1999) Software for use: a practical guide to the models and methods of usage-centered design. ACM Press, New York
3. Cockburn A (2001) Writing effective use cases. Addison-Wesley, Boston
4. Rumbaugh J (1994) Getting started: using use cases to capture requirements. J Object Orient Prog 7(5):8–23
5. Kulak D, Guiney E (2000) Use cases: requirements in context. ACM Press, New York
6. Weidenhaupt K et al (1998) Scenario usage in system development: a report on current practice. IEEE Software 15(2):34–45
7. Arlow J (1998) Use cases, UML visual modelling and the trivialisation of business requirements. Req Eng 3(2):150–152
8. Lilly S (1999) Use case pitfalls: top 10 problems from real projects using use cases. In: Proceedings of TOOLS USA 1999, IEEE Computer Society, Santa Barbara, California
9. Anton AI et al (2001) Deriving goals from a use case based requirements specification. Req Eng 6(1):63–73
10. CCIMB (1999) Common criteria for information technology security evaluation. Technical report, CCIMB-99–031, Common Criteria Implementation Board
11. ECMA (1999) ECMA protection profile: E-COFC public business class. Technical report, TR/78, ECMA International, Geneva, Switzerland
12. Crook R et al (2002) Security requirements engineering: when anti-requirements hit the fan. In: Proceedings of the 10th anniversary IEEE international requirements engineering conference (RE'02), Essen, Germany
13. Pohl K (1994) The three dimensions of requirements engineering: a framework and its applications. Inform Syst 19(3):243–258
14. Loucopoulos P, Karakostas V (1995) Systems requirements engineering. McGraw-Hill, London
15. Kotonya G, Sommerville I (1997) Requirements engineering: processes and techniques. Wiley, Chichester
16. Mylopoulos J, Chung L, Yu E (1999) From object-oriented to goal-oriented requirements analysis. Commun ACM 42(1):31–37
17. Sindre G, Opdahl AL (2000) Eliciting security requirements by misuse cases. In: Proceedings of TOOLS Pacific 2000, Sydney, Australia
18. Sindre G, Opdahl AL (2001) Templates for misuse case description. In: Proceedings of the 7th international workshop on requirements engineering: foundation for software quality (REFSQ'01), Interlaken, Switzerland
19. McDermott J, Fox C (1999) Using abuse case models for security requirements analysis. In: Proceedings of the 15th annual computer security applications conference (ACSAC'99), Phoenix, Arizona
20. McDermott J (2001) Abuse-case-based assurance arguments. In: Proceedings of the 17th annual computer security applications conference (ACSAC'01), New Orleans, Los Angeles
21. Potts C (2001) Scenario noir (panel statement, p 2). In: Proceedings of the symposium on requirements engineering for information security (SREIS'01), Indianapolis
22. Alexander IF (2002) Initial industrial experience of misuse cases in trade-off analysis. In: Proceedings of the 10th anniversary IEEE international requirements engineering conference (RE'02), Essen, Germany
23. Alexander IF (2002) Modelling the interplay of conflicting goals with use and misuse cases. In: Proceedings of the 8th international workshop on requirements engineering: foundation for software quality (REFSQ'02), Essen, Germany
24. Alexander IF (2003) Misuse cases, use cases with hostile intent. IEEE Software 20(1):58–66
25. Ellison R et al (1999) Survivable network system analysis: a case study. IEEE Software 16(4):70–77

26. Firesmith D (2003) Security use cases. J Object Tech 2(3):53–64
27. OMG (2003) Unified modeling language, version 1.5. Object Management Group, Inc. http://www.uml.org. Cited 21 Nov 2003
28. Sindre G, Opdahl AL, Breivik GF (2002) Generalization/specialization as a structuring mechanism for misuse cases. In: Proceedings of the 2nd symposium on requirements engineering for information security (SREIS'02), Raleigh, North Carolina
29. Kruchten P (2000) The rational unified process—an introduction. Addison-Wesley, Boston
30. Sindre G, Firesmith D, Opdahl AL (2003) A reuse-based approach to determining security requirements. In: Proceedings of the 9th international workshop on requirements engineering: foundation for software quality (REFSQ'03), Klagenfurt, Austria
31. Viega J, McGraw G (2002) Building secure software: how to avoid security problems the right way. Addison-Wesley, Boston
32. Andress M (2002) Surviving security: how to integrate people, process, and technology. Sams Publishing, Indianapolis
33. Devanbu PT, Stubblebine S (2000) Software engineering for security: a roadmap. In: Proceedings of the 22nd international conference on software engineering (ICSE 2000), future of software engineering track, Limerick, Ireland
34. Carroll JM, Swatman PA (1999) Managing the RE process: lessons from commercial practice. In: Proceedings of the 5th international workshop on requirements engineering: foundations of software quality (REFSQ'99), Heidelberg, Germany
35. den Braber F et al (2002) Model-based risk management using UML and UP. In: Proceedings of the 13th IRMA international conference: issues and trends of information technology management in contemporary organizations (IRMA'2002), Seattle, Washington
36. Houmb S-H et al (2002) Towards a UML profile for model-based risk assessment. In: Proceedings of the UML'2002 satellite workshop on critical systems development with UML (CSD-UML'02), Dresden, Germany
37. Breivik GF (2002) Abstract misuse patterns—a new approach to security requirements. Masters thesis, Department of Information Science, University of Bergen
38. OWASP (2001) Application security attack components. The open web application security project. http://www.owasp.org/asac/. Cited 21 Sept 2002
39. Hickey A, Davis AM (2003) Elicitation technique selection: how do experts do it? In: Proceedings of the 11th IEEE international requirements engineering conference (RE'03), Monterey, California
40. Coughlan J, Macredie RD (2002) Effective communication in requirements elicitation: a comparison of methodologies. Req Eng 7:47–60
41. Potts C (1995) Using schematic scenarios to understand user needs. In: Proceedings of the ACM symposium on designing interactive systems: processes, practices, and techniques (DIS'95), Ann Arbor, Michigan
42. Anton AI, Earp JB (2000) Strategies for developing policies and requirements for secure electronic commerce systems. In: Proceedings of the 1st ACM workshop on security and privacy in e-commerce, Athens, Greece
43. van Lamsweerde A, Letier E (2000) Handling obstacles in goal-oriented requirements engineering. IEEE T Software Eng 26(10):978–1005
44. Maiden NAM et al (1998) CREWS-SAVRE: systematic scenario generation and use. In: Proceedings of the 3rd IEEE international conference on requirements engineering (ICRE'98), Colorado Springs, Colorado
45. Rolland C, Souveyet C, Achour-Salinesi CB (1998) Guiding goal models using scenarios. IEEE T Software Eng 24(12):1055–1071
46. Achour-Salinesi CB et al (1999) Guiding use case authoring: results from an empirical study. In: Proceedings of the 4th international symposium on requirements engineering (RE'99), Limerick, Ireland
47. Abrahamsson P et al (2003) New directions on agile methods: a comparative analysis. In: Proceedings of the 25th international conference on software engineering (ICSE'03), Portland, Oregon
48. Liu L et al (2003) Security and privacy requirements analysis within a social setting. In: Proceedings IEEE international conference on requirements engineering (RE'03), Monterey, California
49. Amoroso EJ (1994) Fundamentals of computer security technology. Prentice-Hall, Englewood Cliffs
50. Schneier B (2000) Secrets and lies: digital security in a networked world. Wiley, Chichester
51. Moberg F (2000) Security analysis of an information system using an attack tree-based methodology. Masters thesis, Chalmers University of Technology
52. Chung L et al (2000) Non-functional requirements in software engineering. Kluwer, Boston
53. Lutz RR (2000) Software engineering for safety: a roadmap. In: Finkelstein A (ed) The future of software engineering, ACM Press, New York