University of Applied Sciences Wedel
Department: Computer Science

Term Paper: IT-Security

Misuse Cases / Abuse Cases

Business process modelling and security requirements used in the software development industry

Submitted by: *Alexander Kirtzel*

Matr.Nr.: winf1032

Email: winf1032@fh-wedel.de

Submitted at: *17/11/2014*

Advisor: *Prof. Dr. Beuster*

Contents

Introduction

Electronic devices such as computers and mobile phones have become part of our daily life. We are using them in the private sector as well as professional. We work with lot of data including sensitive and personal information. The data can be stored locally or in clouds on different computer and synchronized between many devices.

There are various types, models and ways to create software. The international standard ISO/IEC 12207 defines all possible tasks of the whole software life-cycle processes including the development as well as maintenance. [1] At the beginning of the most used software development life-cycles (SDLC) methods is the determination of the system's requirements, even when incremental or spiral software development methodology is used. The right understanding and definition is very important for the following steps. It prevents possible additional costs and saves expenses. One possibility to model and visualize the design of a system is the Unified Modeling Language (UML). Besides structured or relationship orientated concepts of UML the use case diagram can be used to describe the behaviour and interactions between an actor and the system.

"[…] requirement engineers tend to focus almost exclusively on functional requirements and largely ignore the so-called non-functional requirements, such as data, interface, and quality requirements, as well as technical constraints. Unfortunately, this myopia means that requirements engineers overlook critically important, architecturally-significant, quality requirements that specify minimum acceptable amounts of qualities, such as availability, interoperability, performance, portability, reliability, safety, security, and usability" [2]

Scope of study

In this article I will give an introduction and review of both misuse cases and abuse cases. By presenting both the basics of software development processes and its im-

portance of security requirements I create awareness as well as a basic understanding. We will learn how to classify the features modelling techniques and the difference between them. As the creators of misuse cases and abuse cases suggest a strict method to develop these process I will introduce the basic components and explain each creation step a little bit further. The final examples should lead to a common understanding and the knowledge how to develop misuse cases and abuse cases by your own.

Software development process

The way of software creation can vary in many cases. A huge impact on the total process is the choice of the right approach applied to software development methodology. Different approaches like sequential development with its waterfall model or the iteratively repeating spiral model as well as rapid agile methods like extreme programming [3] or SCRUM [4] have one thing in common: Requirements are always at the beginning (cf. Figure 1: Software development lifecycle) of the process. Part of the requirements is the design of the security. And its definition is having a huge influence on the whole project.

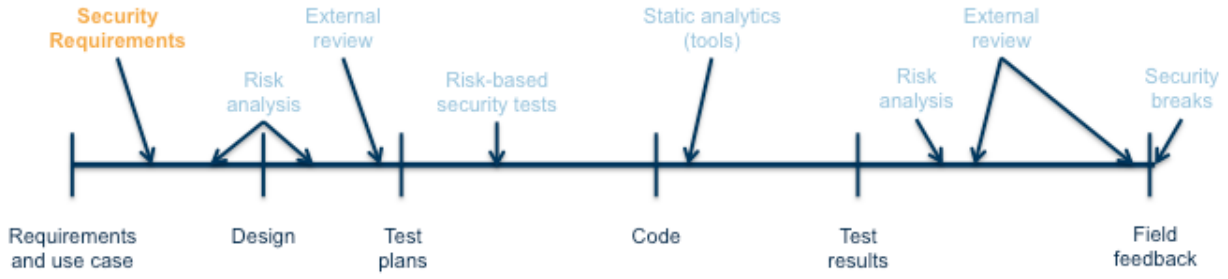

Figure 1: Software development lifecycle [5]

Software Design

In general the software design is known as a process by which an agent creates a specification of a software artefact/model. A modelling language is used to express information or knowledge or systems in a structure that is defined by a consistent set of rules. The rules are used for interpretation of the meaning of components in the structure. Those modelling languages are mainly divided into graphical or textual once. In addition there are some other languages like behaviour driven development, object-oriented or discipline specific.

Graphical	Textual	
• **Unified Modeling Language (UML)** • **Business Process Modeling Notation (BPMN)** • Entity Relationship Model (ERM) • Flowchart • *Many more*	• Formal languages • Gellish • Backus–Naur Form • *Some more*	**Others**

Table 1: Types of modelling languages

Both UML and BPMN are very famous modelling languages. Use cases are part of UML and are used to describe misuse cases as well as abuse cases. Therefore it is important to understand the basics of use cases.

Security requirements

The idea behind the software requirements is the correct engineering of software so that it continues to function correctly under attack. Those security specifications are non-functional requirement to be implemented. Possible attacks may violate the integrity, availability as well as confidential information and lead to a misuse of data and resources.

Until now, no single software engineering methodology exists that can ensure security in the development of large-scale software systems by virtue of nearly infinite ways of attack or misuse. Today's methods aim to mitigate possible risks as much as possible.

Use cases are suitable for most functional requirements, but may lead to neglect of extra-functional requirements, such as security requirements as they are not part of displayed cases.

Business process modelling

Threat model

Use case modelling focus on the definition of functional requirements. Threat models aim to define a set of possible attacks for pieces of software. Those are non-functional requirements which have to be considered during the development process is the goal of threat models. Besides the definition most threat models also evaluate the probability of those cases as well as the potential harm each one may cause, the priority to rank them etc. Those strategies aim to soften or totally eliminate the threats.

Two ways to detect are on the one hand the misuse cases and on the other hand the abuse cases. In short they can be defined as followed:

> ***Misuse Cases****: A process of executing a malicious act against a system. It‘s derived from and the inverse of a use case.*
>
> ***Abuse Cases****: Type of complete interaction between a system and one or more actors, where the results of the interaction are harmful.*

Both – like any threat model – are not a replacement for use cases as mentioned above. They are extending the standardized Unified Modelling Language (UML) notation for use cases to describe different, important scenarios. The purpose is to elect security requirements more easy and to accomplish and understand the software.

Before these threat models there were traditional mathematical security models. But the problem was that they were not easily understandable. Misuse cases and abuse cases

are both using the advantages of clarity and other success factors to develop more secure software.

Use Case

The aforementioned use case diagram is a representation of actors of a system's use case. It describes possible scenarios for a single task or goal. A use case typically describes some function that the system should be able to perform. The description is good for functional requirements (eliciting, communicating and documenting), but not necessarily with extra-functional ones, such as security. Use cases focus on what a system does rather than how the system does it.

According to a known article by Alistair Cockburn – one of the initiators of the agile movement in software development – we know that software development projects where the analysis/requirement phase concentrates primary on use cases rather than textual requirements may be more successful in capturing the users needs. [6] Even if this article may be old compared to the rapid development there are no new methods or contradictory statements. And the application of use cases is also a common practise in agile development projects. [7]

Use cases are lists of steps, typically defining interactions between a role and a system, to achieve a goal. They are describing a user's interaction with the system. There are many other specifications for structural and behavioural UML diagrams. [8] The interactions of cause and effect specify exactly when and under what conditions certain behaviours occur. To make this as easy as possible graphical components are used with as little text as possible. Since the first publication of Ivar Jacobson in 1992 [9] the way to write the content has changed due to many suggestions and experiences of well-known software engineers like Alistair Cockburn, Martin Fowler and Ivar Jacobson himself. In 2011 Ivar Jacobsen et al. published a new version [10] with some changed recommen-

dations, based on their experiences trough many projects. Since this term paper focus on extensions of use cases, we will not discuss the right formal methods. In general an interaction typically consists of an actor, an association and the expected use case (cf. Figure 2 Basic UML components). The actor can be a human/user, an external system or event.

Figure 2: Basic UML components

Misuse Case

In short a misuse case describes a “special kind of use case, describing behaviour that the system/entity owner does not want to occur”. [11] A misuse case defines a process of executing a malicious act against a system. It is derived as a Conceptual extension from and the inverse of a use case, namely “misuse cases”. They are completed sequence of actions, which results in loss for the organization or some specific stakeholder. Most of them may be highly specific situations but also as well as continually threaten systems.

Use cases and misuse cases are illustrated in the same diagram, showing in an “inverted” format. A new element is the misuser: an actor that initiates misuse cases, either intentionally or inadvertently. There are two kinds of associations between a misuse case and a use case.

> ***Threatens**: A regular associated use case of an actor can be threatened by a Misactor to cause harm.*
>
> ***Mitigates**: To prevent or mitigate possible misuse cases a second use case has to be initiated by an actor.*

A use case can mitigate a misuse case. The total goals are to prevent a threat from occurring or if possible to mitigate the impact.

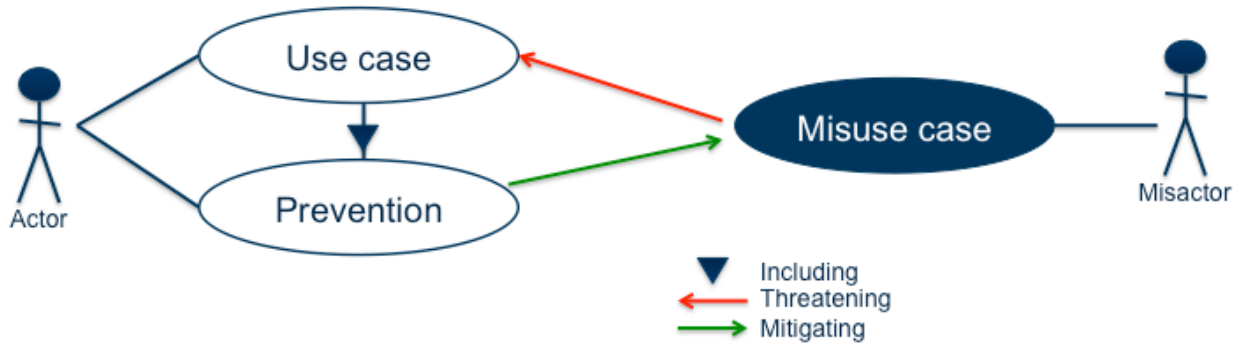

Figure 3: Basic components of a misuse case

Abuse Case

An abuse case is defined as “a specification of a type of complete interaction between a system and one or more actors, where the results of the interaction are harmful to the system, one of the actors, or one of the stakeholders in the system.” [12] The abuse case extends the UML notation but without new terminology or special symbols. They are drawn with the same symbols as a use case diagram. But instead of interacting with the regular cases and to distinguish the diagrams they are kept separate. They describe a type of complete interaction between a system and one or more actors, where the results of the interaction are harmful. One of the main reasons is the abuse of privileges used to complete the abuse case. Strictly speaking an abuse case is a use case until harm caused. Actors are only defined briefly in use cases, but within abuse cases there is a more detailed view on the bad actor and his resources, skills, and objectives for a better understanding.

Figure 4: Basic abuse case components

Comparison of misuse case and abuse case

Both approaches start off by constructing a use case diagram for the scenario. While the misuse case introduce the misuse cases itself, the misactors and all counteracting actions for mitigation, the abuse case shows only the abuse of a system and identifies the actors.

> ***Abuse cases**: Where the results of the interaction are harmful to the system. They are drawn separately from use cases.*
>
> ***Misuse cases**: Behaviour that the system/entity owner does not want to occur. They appear alongside the use cases and there are associations between them.*

To get a final understanding of each modelling language and its goal, the following modified example, based on the abuse description from John McDermott and Chris Fox will help: [12]

> **Situation**: A user forgot to logout of a social network on a public computer
>
> A misuse case might suggest to automatically logout after 5 minutes of inactivity to threaten the chance of possible misuse by a following user on that computer. This interaction is not an abuse case, simply because no harm has been created, yet. No actor has used the login to reveal contents of a private message or make unauthorized changes to the profile. Only when the actor posts private information or access private data, an abuse case takes place.
>
> The definition of a misuse case however, refers to behaviour. Even though no harm resulted because no one used the computer within the 5 minutes, the fact

that a user forgot to logout would result in a misuse case, as it is an unwanted situation.

Methodology

The process of creation possible threat models requires creativity, empathy and knowledge. Therefore the recommended best practise methods, which will help to identify possible attacks, will be explained.

Misuse Case

Building misuse cases is an alternating process repeated for each use case on its own. It is recursively switching from system to subsystem levels or lower if necessary. For each new threatening attack a new mitigating prevention appears. The lower-level cases can highlight aspects, which were not considered at higher levels. The initially top-down process consists of the main aspects to identify, study, prototype, evaluate, and select mitigation approaches. By simply drawing the agents and cases explicitly, its simplicity helps to focus the attention on the elements of the scenario. The process modellers do not have to spend time thinking about the right syntax usage or deeper dependencies. Finished misuse-cases inform developers about which security-related information they should specify and not about how and when to do so.

Developing process

Figure 5: Misuse case developing process

This 5-step process will repeat after each pass until a necessary security level is reached. It is recommended to include this security requirements process in the development process. The following list gives a more detailed overview of each step: [11]

1. **Identify critical assets**: Assets can either be information that an enterprise possesses, virtual locations that the enterprise controls or computerized activities that the enterprise performs.
2. **Define security goals**: Preferably by using a standard typology of security goals for each asset.
3. **Identify threats**: For each previous defined security goal threats can be identified by focussing on two aspects:
    a. Stakeholders that may intentionally harm the system
    b. Sequences of actions that may result in intentional harm
4. **Identify and analyse risks**: For each threat by using standard techniques for risk analysis as well as calculating the costing from the security and safety engineering fields
5. **Define security requirements**: Threats should match the risks and protection costs, preferably aided by a taxonomy of security requirements

Abuse Case

A structured approach requires to analyse each possible target/component. Using a tree diagram (cf. Figure 6: Example of a tree diagram) is helpful to evaluate each component. Starting with the modelled system as the root element itself and with all components and resources as leaves and finally followed by interior nodes like the sub-systems, applications and individual classes. The trees are similar to those used in penetration testing and attack trees. [12] Each abuse case includes a description of the range of security privileges that may be abused and are used for following actions as well as for assessing the harm that results from an abuse case.

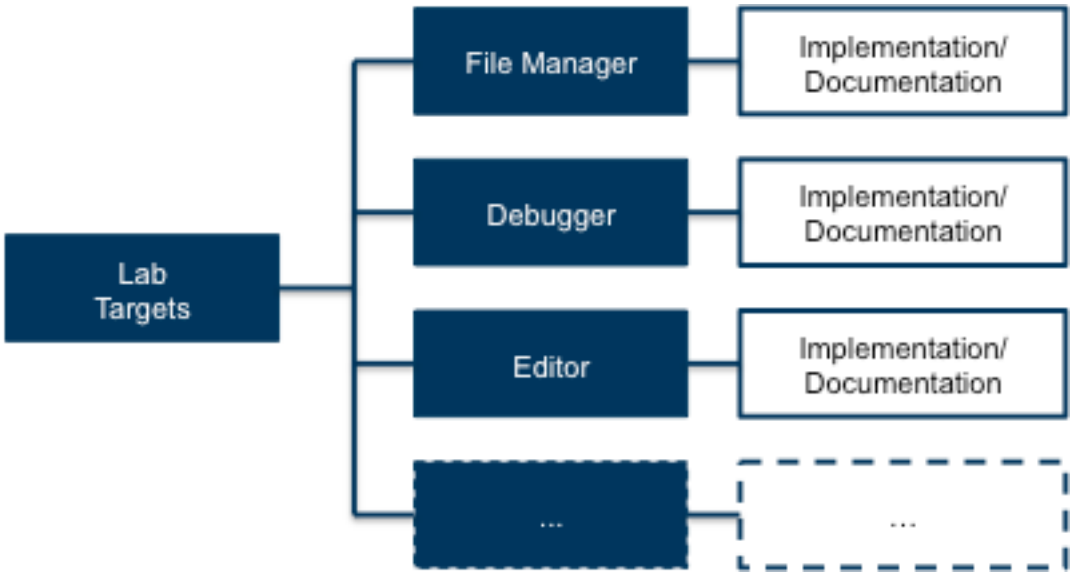

Figure 6: Example of a tree diagram

Developing process

The abuse model is usually developed one step behind the use case model. Each component of the use case model is used to construct the corresponding component of the abuse case model. This offers a more structured proceed than just simply guessing possible abuses. Like the misuse cases approach it is a five-step process, too. [12]

Figure 7: Abuse case developing process

But instead of the instant repetition to explore mitigating strategies against the previews threat, an abuse model acts mostly independent.

1. **Identify the actors**: For each harmless actor a malicious actor will be added. All regular actors can found in the requirement documents. In addition to identify further malicious actors a carful analysis of the system environment is useful.

2. **Identify the abuse cases**: Each interaction with the system has to be identified. For the following steps the abuse case gets a name and will be created using basic UML components.
3. **Define abuse cases**: Refine the description of the system.
4. **Check granularity**: The engineer has to make sure to follow the cost-benefits. Therefore he has to be aware of
    a. Including possible but unlikely cases
    b. Modelling with too much detail

    Deciding how many abuse cases are needed is largely a matter of experience and consideration of the specific target system.
5. **Check completeness and minimality**: The goal is to check if a critical abuse case may have been omitted. A review of requirement documents and use case model might be helpful. In addition the engineer may consult users and customers to be sure that no critical abuse has been overlooked.

Cases/Scenarios

After explaining the basics concepts and the suggested ways to develop misuse cases and abuse cases, the following examples for both will show a basic usage. None of the following examples is a complete model for the corresponding situation.

Misuse case

The following examples are based on Ian Alexander. [13] The scenario is about how to ensure that a driver can drive his own car.

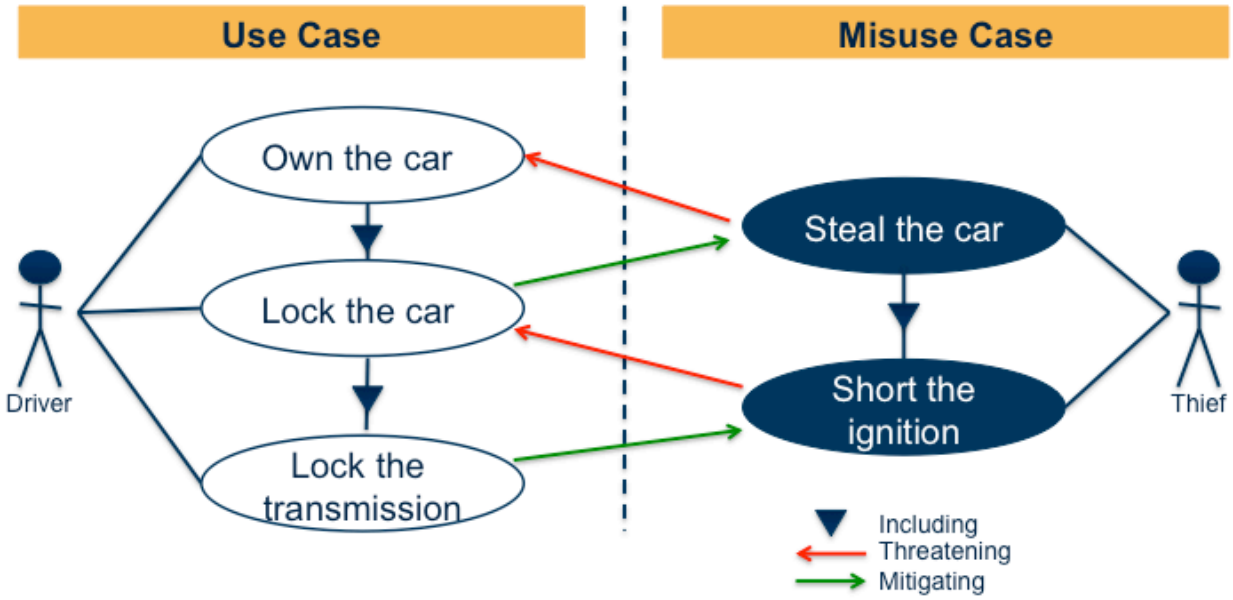

Figure 8: Example of a misuse case with a human actor [13]

Figure 8 shows an example of a misuse case with a human actor. The threatening and mitigating strategies are a balanced zigzag pattern of play and counter play. To develop this example the security requirements process of misuse cases will be repeated. The first use case is that a driver owns a car. In this case the car is the main object and required to drive. A goal is to make sure that this car is not stolen. This leads to the first possible threat: A thief might steal the car. This is not an unusual event and the driver has to take precautions by simply locking the car.

During the second repetition of this case a new threatening action is identified. A thief could short the ignition. To mitigate this the transmission could be locked. This might be a more expensive precaution than just locking the car but during step four and five of the development process the engineer might decide that the value of the car is worth it to protect by more effort.

A safety requirement scenario does not necessarily involve a human agent (cf. Figure 9: Example of a misuse case with a non-human actor). The second example defines bad weather as a potential risk.

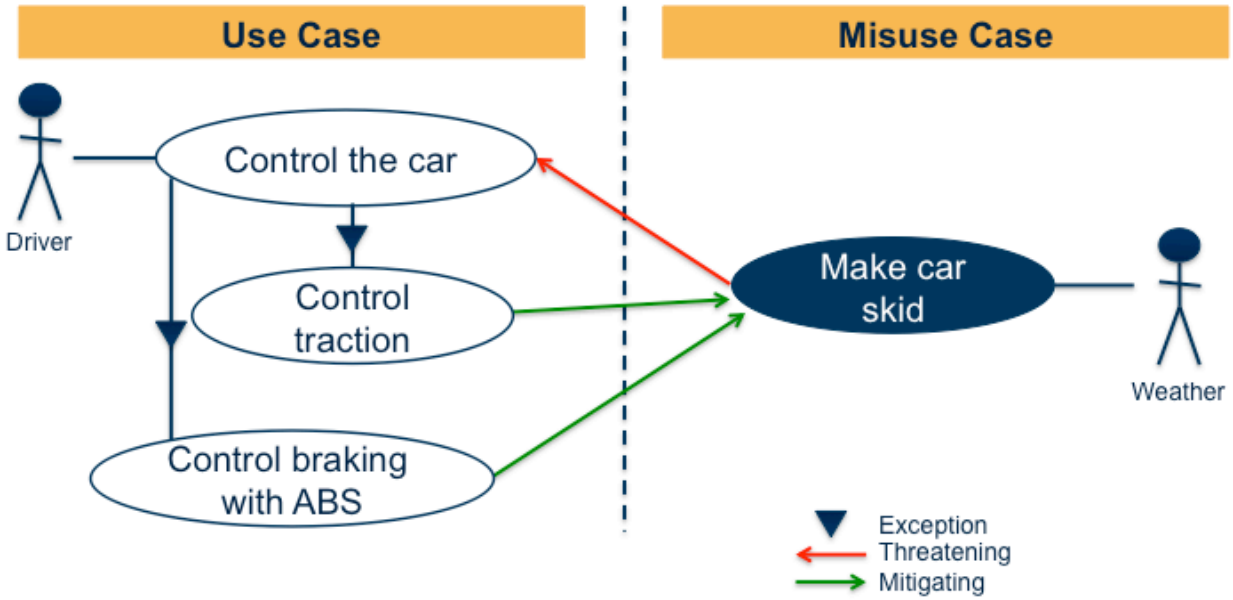

Figure 9: Example of a misuse case with a non-human actor [13]

Usually the driver controls the car while driving. Due to heavy rainfall or slippery streets the car can skid. In most cases the driver could mitigate by controlling the traction. A second iteration leads to the mitigating action to control braking with the ABS-system. This example shows that there could be multiple ways to mitigate a threatening case.

Discussion

One of the strengths of misuse cases is the early focus on security by describing security threats and then requirements, without going into design. Switching to the misuser perspective increases the chance of discovering threats. The simplicity of legibility of those diagrams helps stakeholders to understand the importance of security, too. Ordering and estimating the consequences helps to prioritize the requirements. A well-documented library helps by tracing a lack of security requirements and most of the dia-

grams are created on a generic level, which helps to easily reuse them for following or different projects.

On the other hand there are also some not negligible weaknesses with misuse cases. The open-ended method guidelines mean that developers will have to improve their knowledge about the development process. There might be a potentially large number of threats that must be considered. This may lead to analysis paralysis. While also those misuse cases do not always follow an identifiable sequence of actions. As afore-mentioned it requires some experience to follow the cost benefit criteria.

Abuse case

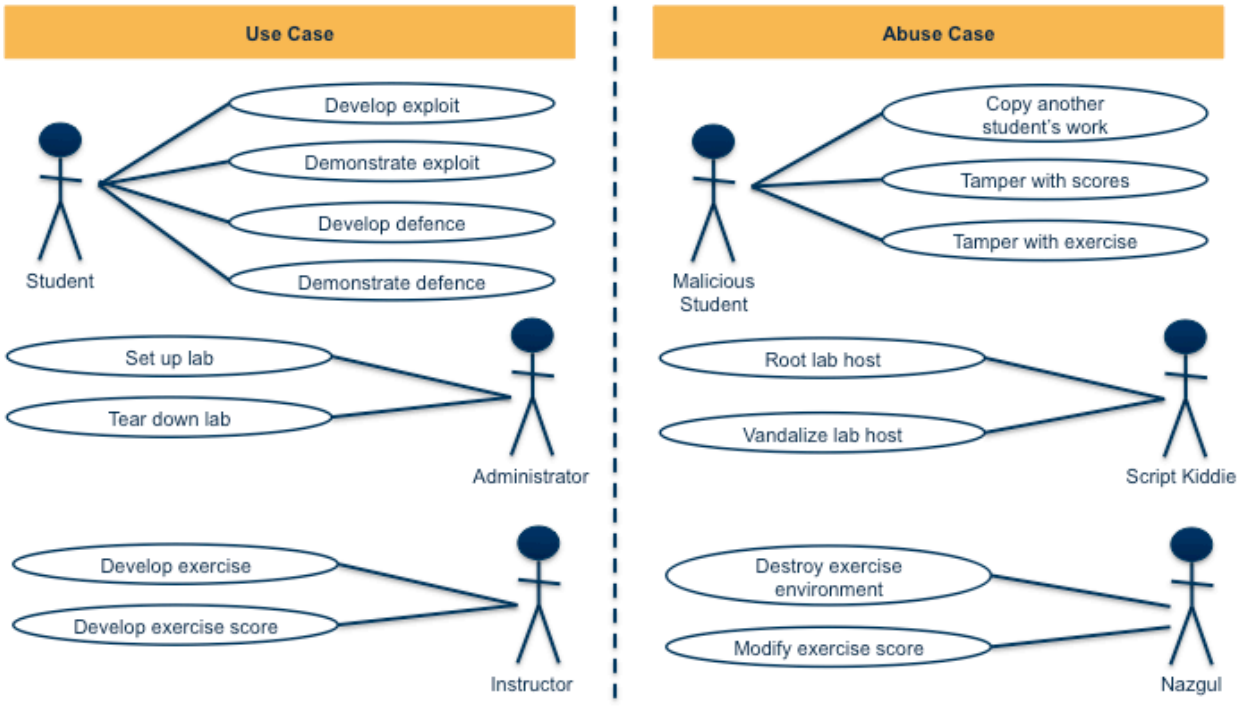

Figure 10: Example of abuse cases with corresponding use cases Modified example [14]

As shown in Figure 10: Example of abuse cases with corresponding use cases on the left side there are some use cases, describing regular and desired actions. Students

first develop exploits and demonstrate it later. Subsequently they develop a defence strategy and demonstrate them again. That describes a good student. Following the developing process for abuse cases the first step of identifying the actor introduces the malicious student. As known from the attack tree one case would be the development and demonstration of both exploit and defence. An abuse case of the malicious student might be to copy another student's work unasked. Since primary school people know that this could be a possible attack which needs to be addressed. A new iteration of the process leads to the possible abuse case that a malicious student might tamper with scores. After consulting other teachers the tampering with exercises is identified as a new abuse case.

Discussion

As abuse cases are developed as independent diagrams they are refuting to characterize the assurance. Most known attacks can be described individually, thus they can be excluded as potential risk. The formal methods to proceed often help to overcome a lack of creativity. Abuse cases can be ranked or weighted according to the assurance that should be applied to them. The assurance budget for a project can then be allocated by abuse case, according to the ranking.

Even if the formal method helps to precede a great creative imagination and empathy is still a premise to cover most risks. The engineer still has to assume events a person usually cannot or will not do. And even if there are many interfaces it requires a careful look at all of them including environmental factors.

Conclusion

It is difficult if not impossible to find a balance between over-engineering and harmful lacks of security in general. This requires experience. Each development needs its own requirements and requires engineering experiences. Another criteria might be the cost-benefit. Spending too much effort with unlikely or incidental cases with only a little im-

pact on the actual security might be unnecessary. As mentioned before the experience with security know-how as well as subject matter expertise plays a key role in identifying and prioritizing threats. As one security gap may lead to unwanted consequences it is difficult to strike the right balance between cost and value.

In most cases the mitigation measures do not neutralize all possible security threats. Thieves may pick unsuspected access paths. But sometimes as secure counting measures become unsecure, leading to new ways of exploring a system. [15, 16] However, partial mitigations can still be useful as long as they afford a realistic increase in protection at reasonable cost. Neutralizing all possible threats is wishful thinking and cannot be a requirement for any software.

The aforementioned strategies are good to identify threats and useful for security requirements elicitation and recommended by Guttorm Sindre and Andreas L. Opdahl [11], the founders of misuse cases and John McDermott and Chris Fox who defined the abuse cases [12]. Sometimes devising threats and negative agents can be a more powerful technique. In general the following questions help to identify threats:

Who might want this to go wrong?

What could they do to make this go wrong?

The step of identifying the threats helps both engineers and customers to increase the understanding of the security features. By using use cases and its extensions engineers are able to immediate justify explicit known threats by knowing where to start and because of to its easy and fast to understand visual representation.

As said many times before: there is not an ideal approach of how to develop software the best way. Tools and models will just improve that process. Unfortunately there are no real representative evaluations of both misuse cases and abuse cases in practical software development projects. Therefore one cannot say which approach is better. But

according to the search results and written papers, people are tending to prefer misuse cases.

It needs to be said that most of the work presented here is either based on Guttorm Sindre and Andreas L. Opdahl main article “Eliciting security requirements by misuse cases” and their following paper “A Reuse-Based Approach to Determining Security Requirements” when talking about misuse cases. [11, 17] Or based on the paper “Using Abuse Case Models for Security Requirements Analysis” by John Pierce McDermott and Chris Fox when talking about abuse cases. [12] Most of the additional literature is attributable to both papers.

References

[1] International Standard ISO/IEC 12207:2008 (2013), Systems and software engineering - Software life cycle processes
http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=43447

[2] D. Firesmith (2011), The Importance of Safety- and Security-related Requirements, First of a Three-Part Series
http://blog.sei.cmu.edu/post.cfm/the-importance-of-safety-and-security-related-requirements

[3] A. Cockburn (2006), Agile Software Development: The Cooperative Game

[4] N.S. Janoff and L. Rising (2000), The Scrum Software Development Process for Small Teams

[5] P. Hope and G. McGraw (2004) Misuse and Abuse Cases: Getting Past the Positive

[6] A. Cockburn (2008), Why I still use use cases
http://alistair.cockburn.us/Why+I+still+use+use+cases

[7] B. Linders (2014), Applying Use Cases in Agile: Use Case 2.0, Slicing and Laminating
http://www.infoq.com/news/2014/02/use-cases-agile

[8] The Unified Modeling Language
http://www.uml-diagrams.org/

[9] I. Jacobson (1992), Object-Oriented Software Engineering: A Use CASE Approach

[10] I. Jacobson, et. al. (2011), Use Case 2.0: The Guide to Succeeding with Use Cases

[11] G. Sindre and A. L. Opdahl (2004), Eliciting Secutiry Requirements with Misuse Cases

[12] J. McDermott, C. Fox (1999), Using abuse case models for security requirements analysis

[13] I. Alexander (2003), Misuse Cases: Use Cases with Hostile Intent

[14] C. Wei (2005), Misuse Cases and Abuse Cases in Eliciting Security Requirements

[15] The Heartbleed Bug
http://heartbleed.com/

[16] OpenSSL Security Advisory
https://www.openssl.org/news/secadv_20140407.txt

[17] G. Sindre, D. G. Firesmith, A. L. Opdahl (2003), A Reuse-Based Approach to Determining Security Requirements

All references and links were collected in May 2014.