# Visualizing Cyber Attacks with Misuse Case Maps

Peter Karpati[1], Guttorm Sindre[1], and Andreas L. Opdahl[2]

[1] Department of Computer and Information Science
Norwegian University of Science and Technology, NO-7491 Trondheim, Norway
{kpeter,guttors}@idi.ntnu.no
[2] Department of Information Science and Media Studies,
University of Bergen, Bergen, Norway
Andreas.Opdahl@uib.no

**Abstract.** [**Context and motivation**] In the development of secure software, work on requirements and on architecture need to be closely intertwined, because possible threats and the chosen architecture depend on each other mutually. [**Question/problem**] Nevertheless, most security requirement techniques do not take architecture into account. The transition from security requirements to secure architectures is left to security experts and software developers, excluding domain experts and other groups of stakeholders from discussions of threats, vulnerabilities and mitigations in an architectural context. [**Principal idea/results**] The paper introduces *misuse case maps*, a new modelling technique that is the anti-behavioural complement to use case maps. The purpose of the new technique is to visualize how cyber attacks are performed in an architectural context. [**Contribution**] The paper investigates what a misuse case map notation might look like. A preliminary evaluation suggests that misuse case maps may indeed make it easier for less experienced stakeholders to gain an understanding of multi-stage intrusion scenarios.

**Keywords:** security, requirements elicitation, misuse case, use case map, misuse case map.

## 1 Introduction

Much effort in the security area focuses on *surveillance* and *fire-fighting*, which are undoubtedly crucial aspects of the "cops and robbers" game of security. A complementary approach is *prevention by design*. Instead of detecting and mitigating attacks, prevention by design strives to eliminate security vulnerabilities in the early phases of software development. Vulnerabilities can take many shapes. One time, a well-known, long-used mechanism is misused in an unexpected way. Another time, an obscure part of the software system is exposed and exploited by an attacker. In order to eliminate vulnerabilities early during software development, it is essential to understand the attackers' perspectives and their ways of working, as pointed out by many authors (e.g., [1-3]).

Much research has been performed on modelling the technical aspect of complex attacks, targeting security experts and security-focused software developers. Our premise is that secure software development may benefit from involving a wider

R. Wieringa and A. Persson (Eds.): REFSQ 2010, LNCS 6182, pp. 262–275, 2010.
© Springer-Verlag Berlin Heidelberg 2010

group of stakeholders, such as domain experts (who know the subject and usage worlds of the proposed software system) and regular software developers who have no special security training. Ideally, this would happen during the requirements phase of software development, which is a common ground for domain experts, software experts and security experts to meet. Also, clarifying security issues already during the requirements phase results in a security conscious design that saves many troubles (money, effort, time, reputation etc.) later on.

There are already several techniques and methods available for dealing with security requirements in the early software development phases. But there is no technique or method that addresses security requirements in relation to design of secure architectures. In practice, however, the two cannot be completely separated: possible threats will depend on the chosen architecture; the choice of architecture might depend on what threats are considered the most dangerous ones; different architecture choices offer different mitigations strategies etc. Our idea is that domain experts, regular software developers and others should be allowed and encouraged to reason about security concerns in an architectural context. For this purpose, suitable representations are needed that combine user, designer and security perspectives on the proposed software system, so that all stakeholders can understand the issues and contribute their ideas and background knowledge to the security discussions.

Hence, the purpose of this paper is to introduce a new attack modelling technique that combines an attacker's behavioural view of the proposed software system with an architectural view. The technique is intended to be useful for a variety of stakeholders. The technique is called *misuse case maps (MUCM)*. It is inspired by *use case maps* [4, 5], into which it introduces anti-behaviours. The technique is illustrated through a multi-stage intrusion from the literature [6]. Results from a preliminary evaluation are also reported.

The rest of the paper is organized as follows. We present related work in Sec. 2. Misuse case maps are introduced in Sec. 3. Misuse case maps are applied to an example from the literature in Sec. 4. A preliminary evaluation is presented in Sec. 5. Finally, we conclude and point out future directions for our work in Sec. 6.

## 2 Modelling Techniques for Security Requirements

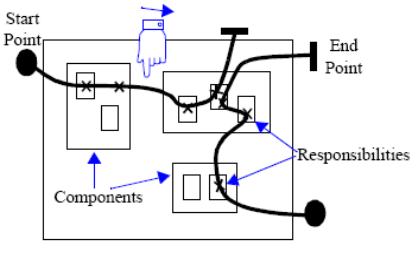
### 2.1 Security Requirements

There are already many techniques and methods available that focus on elicitation and analysis of security requirements during early RE. Attack trees [7] and threat trees [8] are trees with a high level attack (or threat) at the root, which is then decomposed through AND / OR branches. Secure i* [9] is an extension of the i* modelling language, where malicious actors and their goals are modelled with inverted variants of the usual icons. Abuse frames [10], extend problem frames with anti-requirements that might be held by potential attackers. Abuse cases [11], misuse cases [12], and security use cases [13] are security-oriented variants of regular use cases. Abuse and misuse cases represent behaviours that potential attackers want to perform using the software system, whereas security use cases represent countermeasures intended to avoid or repel these attacks. The difference between abuse and misuse cases is that the

latter show use and misuse in the same picture, whereas abuse cases are drawn in separate diagrams. We will return to misuse cases in Sec. 2.4.

There are also techniques and methods that attempt to cover later development phases. Secure Tropos [14] extends the Tropos method with security-related concepts, whereas KAOS has been extended with anti-goals [15]. The CORAS project [16] combined misuse cases with UML-based techniques into a comprehensive method for secure software development. Other security-focused extensions of UML include UMLsec [17] and SecureUML [18]. Languages for secure business process modelling have also been proposed based on BPMN [19] and UML activity diagrams [20]. Security patterns describe recommended designs for security [21], and the formal specification language Z has been used to specify security-critical systems [22, 23].

Despite the many techniques and methods available for dealing with security in the early phases of software development, there is so far no technique or method that links security requirements and architecture. There is, however, a technique that links software functionality in general with architecture, which we now present.




Imagine tracing a path through a system of objects to explain a causal sequence, leaving behind a visual signature. Use Case Maps capture such sequences. They are composed of:

- **start points** (filled circles representing pre-conditions or triggering causes)
- causal chains of **responsibilities** (crosses, representing actions, tasks, or functions to be performed)
- and **end points** (bars representing post-conditions or resulting effects).

The responsibilities can be bound to **components**, which are the entities or objects composing the system.

**Fig. 1.** Notation and interpretation of UCMs (from [4])

## 2.2 Use Case Maps (UCM)

The *use case map (UCM)* notation [4,5,24,25] was introduced by Buhr and his team at Carleton University in 1992. It quickly gained popularity. UCMs have been used in both research and industry, in particular in the telecommunications sector. It is a part of the User Requirements Notation (URN) standardized by the International Telecommunication Union (ITU).

UCMs provide a combined overview of a software system's architecture and its behaviour by drawing usage *scenarios paths* (aka use cases) as lines across boxes that represent architectural run-time *components*. The boxes can be nested to indicate hierarchies of components. The scenario paths are connected to the components they run across by *responsibilities* drawn as crosses. Fig. 1 illustrates and explains the basic UCM notation. This UCM shows multiple scenarios as multiple paths across the architecture components.

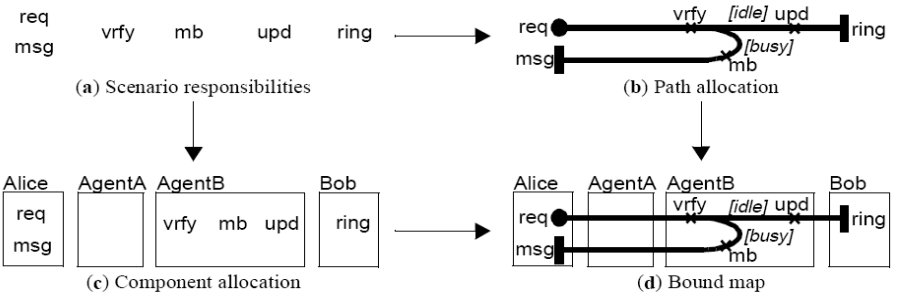



**Fig. 2.** Variants of UCMs (from [4])

Fig. 2 shows how a UCM binds responsibilities, paths, and components together. In this simple example, "…a user (**Alice**) attempts to call another user (**Bob**) through some network of agents. Each user has an agent responsible for managing subscribed telephony features such as Originating Call Screening (**OCS**). Alice first sends a connection request (**req**) to the network through her agent. This request causes the called agent to verify (**vrfy**) whether the called party is idle or busy (conditions are between square brackets). If he is, then there will be some status update (**upd**) and a ring signal will be activated on Bob's side (**ring**). Otherwise, a message stating that Bob is not available will be prepared (**mb**) and sent back to Alice (**msg**)." [4] The example also shows how sections of scenario paths can be split and joined to indicate alternative or parallel paths.

In this manner, UCMs offer high-level views for software and systems development [4, 24]. They combine an architectural overview with behavioural detail and thus facilitate discovery of problems within collections of scenarios or use cases. UCMs can also serve as synchronization means among the scenarios/use cases to check them for completeness, correctness, consistency, ambiguity or consistent abstraction levels. UCMs provide several additional notations for visualizing more complex behaviours and more refined relationships between scenarios and architecture components. We do not present all of them here. The UCM notation also offers variants that use only two of the three core components (scenario paths, architecture components and responsibilities) [4, 25]. In particular, paths and components can be used without responsibilities for presenting very-high level overviews and for "napkin-type" sketching of ideas.

### 2.3 Use Case Maps and Anti-functional Requirements

Security has been discussed in relation to UCMs in connection with *performance-related completions* ("additions, including annotations, component insertions, environment infrastructure, deployment, communication patterns, design refinements and scenario or design transformations which correspond to a given deployment style") [26] and in connection with *RE for data sharing in health care* in [27]. UCMs are used to represent an example of *early aspects* at the requirements level in [28]. Security appears there as a MUCM component, first in the main UCM and later in a plug-in (or sub-map) of the main UCM. However the aim of the example is to demonstrate the UCM notation and not to address security as such.

Beyond these contributions, we have not found any direct considerations of the *security* perspective in UCMs. But there are two contributions that address *safety* in a UCM context. We review them here because security and safety requirements are both examples of *anti-functional requirements*, i.e., requirements that state what the software system should *not* do. Hence they are similar to functional requirements in that they are both concerned with the software system's behaviour, but they have opposite *modalities*. Wu and Kelly [29] present an approach to derive safety requirements using UCMs. The approach aims to provide assurance on the integrity of requirements elicitation and formulation. First they formulate the problem context in their process, followed by analysis of deviations, assessment of risks, choice of mitigations and formulation of safety requirements. The initial set of requirements is refined iteratively while a software system architecture is also developed incrementally. The authors conclude (1) that UCMs are effective for capturing the existing architectural context (structure and specific operational modes) beside the intended behaviour and (2) that the explicit architectural references extend the scope of the deviation analysis compared with the one over functions or use cases. In [30], Wu and Kelly extend their approach into a negative scenario framework (along with a mitigation action model), which has a wider theoretical background and is more general than the proposal in [29]. The UCM no longer plays the central role, and the approach to identifying deviations is less specific. Although their framework targeted safety-critical systems, the authors suggest that it is applicable for other systems as well, such as security- and performance-critical ones.

Despite the interest in combining UCMs with anti-functional requirements, no representation technique has so far been proposed that provide a combined overview of the attackers' and the architects' views of a proposed software system. However, there is a technique that shows how to extend and combine representations of wanted software system behaviour, as covered by UCMs, with an attacker's attempts to cause harm, which we now present.

### 2.4 Misuse Cases (MUC)

Misuse cases (MUC) [12] extend use cases (UC) for security purposes with *misusers*, *misuse cases* and *mitigation use cases*, as well as the new relations *threatens* and *mitigates*. They represent security issues by expressing the point of view of an attacker [31]. Whereas regular UCs describe functional requirements to the software system, MUCs thereby represent *anti-functional requirements*, i.e., behaviours the software should prohibit. They thus encourage focus on security early during software development by facilitating discussion between different stakeholder groups, such as domain experts, software developers and security experts. MUCs have also been investigated for safety [32-34] and other system dependability threats [35] and compared with other similar techniques like FMEA, Attack Trees and Common Criteria [33, 36, 37]. MUCs can be represented in two ways, either diagrammatically or textually. Diagrammatically, MUC symbols inverts the graphical notations used in regular UC symbols, and UC and MUC symbols can be combined in the same diagram. Textually, both lightweight and an extensive template are offered [12, 34].

# 3 Misuse Case Maps (MUCM)

## 3.1 The Need for Misuse Case Maps

The previous section shows that there are many techniques and methods available for dealing with security requirements in the early software development phases, both during RE and in the transition to later phases. But there is no technique or method that addresses security requirements in relation to design of secure architectures. Yet it is well known that requirements and architecture can rarely be considered in complete isolation. Contrarily, architecture is essential for security in several ways. The types of architecture components suggest typical weaknesses and attack types for the component (e.g., a router can be scanned for open ports). The specifics of architecture components suggest specific weaknesses (e.g., a particular router model is likely to have a particular standard password). The path each function takes through the software architecture suggests which general and specific weaknesses a user of that function might try to exploit. Furthermore, when weaknesses have been identified, architectural considerations are equally important for mitigating the threats. To alleviate these and other problems, there is a need for a security requirements technique that combines an attack-oriented view of the proposed software with an architectural view.

## 3.2 Basic Concepts

MUCMs extend regular UCMs for security purposes with *exploit paths* in much the same way that MUCs extend regular UCs with *misuse cases*. As in regular UCMs, the *exploit paths* in a MUCM are drawn across nested boxes that represent hierarchically-organized architecture *components*. In addition to regular *responsibilities*, the intersection of an exploit path and a component can constitute a *vulnerability*, which is a behavioural or structural weakness in a system. A component can be a *vulnerability* too. A threat combines one or more weaknesses to achieve a malicious intent. Vulnerabilities can be *mitigated*, where a mitigation counters a threat and translates to a security requirement. Both regular scenario paths and exploit paths can be combined in the same MUCM, just like a MUC diagram can also show UCs.

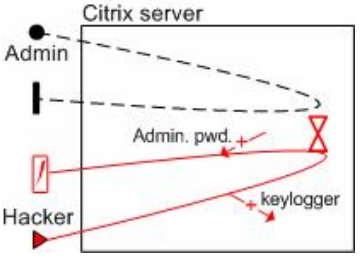



**Fig. 3.** A simple MUCM example

Fig. 3 shows an excerpt of a MUCM comprising one component, a server, along with a regular scenario path and an exploit path. The excerpt is part of the bigger example presented in Sec. 4. We will reveal further details of the notation below.

### 3.3 Notation

The MUCM notation is based on the regular UCM notation, just like the MUC notation is based on the UC notation. The MUC notation uses inversion of use-case symbols to distinguish wanted from unwanted behaviour. This is not easy to do for use case maps, where the start and end points of regular scenario paths are already shown with filled icons and where the paths themselves are drawn as whole lines. Instead, we have explored using colours and different icon shapes to distinguish between wanted and unwanted behaviours in MUCMs.

The leftmost column of Fig. 4 shows the basic MUCM symbols. An exploit path starts with a *triangle* and ends in a *bar* (as in UCMs) if no damage happened. Otherwise the path ends in a *lightning* symbol. Exploit paths can be *numbered* to show the order of stages in a complex intrusion, where the stages will mostly be causally related, in the sense that each of them builds on the results of previous ones. A *vulnerable* responsibility (e.g., an authentication point) or component (e.g., an unpatched server) is indicated by a closed curve, and a *mitigation* of the resulting threat (e.g., secure password generation, routines for patching servers) is shown by shading the interior of the closed curve. *Responsibilities* can be left out whenever they are not relevant from the intrusion's point of view. Through these basic symbols, MUCMs offer a basic notation that is close to the simplified UCM notation suggested for very-high level overviews and for "napkin-type" sketching of ideas. We expect this to be the most prominent use of MUCMs in practice.

Yet at this early stage it is worth exploring more detailed notation alternatives too. For example, the rightmost column of Fig. 4 shows how a time glass can be used to indicate that an exploit path must *wait* for a regular scenario path to reach a certain point. The example in Fig. 3 used this notation to show how an attacker, who have secured access to a Citrix server at an earlier intrusion stage, installs a keylogger on the server in order to snatch the administrator's password. The hour glass indicates that the attacker has to wait for an administrator to log in before the keylogger can snatch the password.

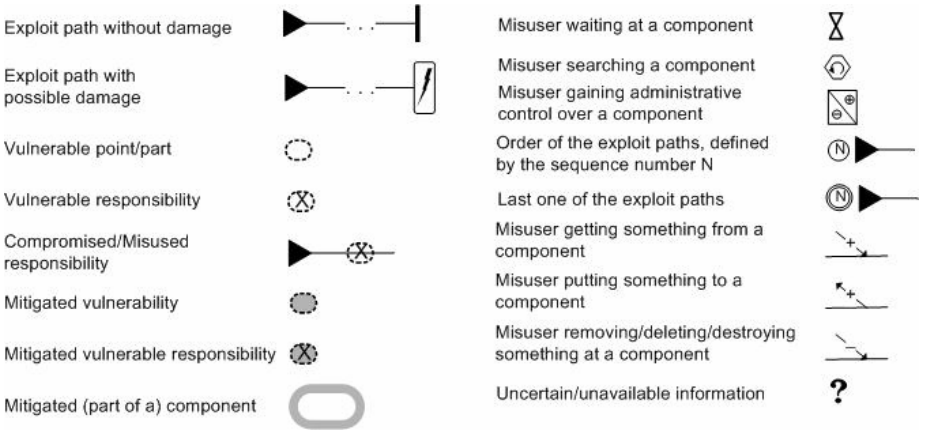



**Fig. 4.** The proposed MUCM notation, with the basic symbols shown on the left and further tentative extensions on the right

*Get*, *put* and *remove* arrows can be used to show how an exploit path interacts with a component. An example involving the *get* arrow is when the attacker accesses the password hash files. An example of a *put* arrow is when the attacker installs a sniffer program on one of the servers. An example using the *remove* arrow is when the attacker deletes his/her traces from a system. We will see in the complex example that not all the information is available about the case. Similarly, when (re)creating an intrusion, some parts of it may be unclear at first. The question marks can be used as reminders about unclear issues.

*Labels* can be attached to symbols. For example, a label at the start of a UCM path might indicate the *role* of the actor if it affects a connected exploit path; a label at the end of an exploit path might be labelled with the *result* of the exploit if it is not clear from the path alone; and *get*, *put* or *remove* arrows can be labelled with the types of data or software that are accessed. These arrows are part of the regular UCM notation as well, where they have a slightly different meaning. Hence, their interpretation depends on the context. Further work should consider using distinct symbols, such as a wave arrow, to differentiate the notations.

Like in regular UCMs, the granularity of the intrusion representation can change by combining or exploding steps. Consider a case where the attacker downloads a file of password hashes from one machine, cracks them in his/her own computer and proceeds to log into another machine with a cracked password. This could be shown as individual steps or as a composite step – a MUCM *stub* – that hide the cracking process and leads the exploit path from the first machine with the hashes to the one logged into with the cracked password.

As already explained, we expect the basic MUCM symbols to be the ones most used in practice, and we expect the notation we suggest here to evolve further. In this paper, however, we will stay with the symbols we have used in the preliminary evaluation to be presented in Sec. 5.

## 4 A Complex Intrusion Example

MUCMs and the associated notation have been developed through a series of modelling exercises using hacker stories from [6]. We have so far developed MUCMs for 5 of the 12 relevant intrusion cases described there, 3 of them in full detail. We present one of them here to illustrate MUCMs and the first version of the associated notation.

### 4.1 The Bank Intrusion Case

The bank intrusion is a multi-stage intrusion presented in [6, Chapter 7]. We suggest following the intrusion on the MUCM in Sec. 4.2 while reading through the intrusion steps.

First, the intruder found an interesting bank by browsing a web site with organizations and IP ranges assigned. Next, he probed for further details about the IP addresses of the bank and found a server that was running Citrix MetaFrame (remote access software). He then scanned other networked computers for the remote access port to Citrix terminal services (port 1494). The attacker knew he might be able to enter the server with no password, as the default Citrix setting is “no password

required". He searched every file on the computer for the word "password" to find the clear text password for the bank's firewall. The attacker then tried to connect to routers and found one with default password. He added a firewall rule allowing incoming connections to port 1723 (VPN).

After successfully authenticating to the VPN service, the attacker's computer was assigned an IP address on the internal network, which was flat, with all systems on a single segment. He discovered a confidential report written by security consultants containing a list of network vulnerabilities. He also found operation manuals to the bank's IBM AS/400 server on a Windows domain server. The default password

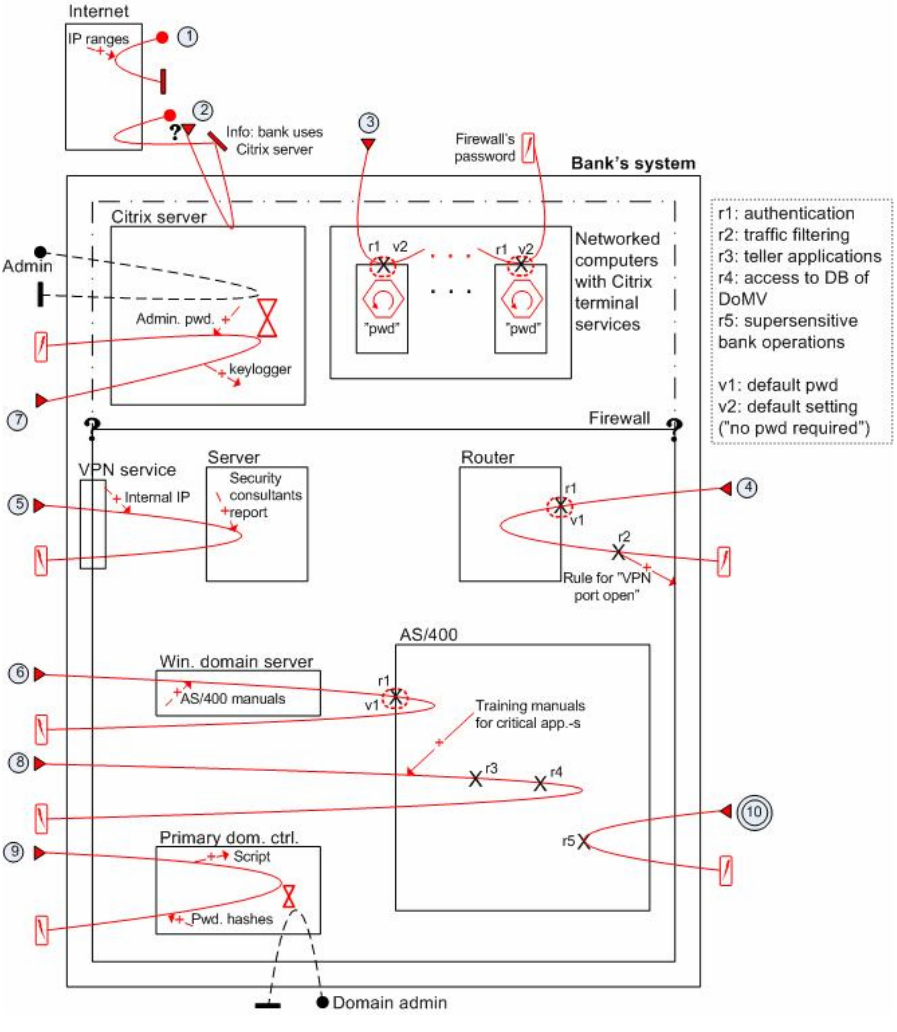



**Fig. 5.** A misuse case map for the bank intrusion. The whole red line depicts the attacker's footprint whereas the dashed black line shows the regular users' activities.

worked for the AS/400. The intruder installed a keylogger on the Citrix server, waited for an administrator to log in and snarfed the administrator's password. He now had access to training manuals for critical AS/400 applications, giving him the ability to perform any activity a teller could. He also found that the database of the Department of Motor Vehicles was accessible from the bank's site. He accessed the Primary Domain Controller (which authenticates login requests to the domain) and added a disguised script that extracted password hashes from a protected part of the system registry in the administrator's startup folder. He then waited for a domain administrator to log in so the script would be triggered and password hashes written to a hidden file. He then cracked the appropriate password. The most sensitive parts of the bank's operations could now be accessed (generating wire transfers, moving funds etc.). A manual he had already found described how to complete a wire transfer form.

The attacker was a "white-hat" hacker who claimed not to harm the bank or its customers as a result of the intrusion.

### 4.2 MUCM

Fig. 5 shows a MUCM for the bank intrusion. Some details were omitted, either because they were not given in the original text (e.g., how the access to some of the components was secured) or because the details were intuitive and would only overload the map (e.g., to access the internal computers, the attacker always went through the VPN).

## 5 Preliminary Evaluation

To preliminary evaluate MUCMs and the MUCM notation, we sent out a written evaluation sheet to more than 20 colleagues and other contacts. We received 12 responses. All respondents had MSc or PhD degrees, except one MSc student who, on the other hand, had professional experience as a system administrator. All the degrees were in computing. 6 of the respondents were working as academics, 4 in industry and 2 in both academia and industry.

The evaluation sheet had three sections. The first section explained the aim and the required conditions of the experiment. There was no time limit, but the respondents were asked to perform the evaluation without interruption. The second section gave an introduction to UCM and MUCM. The third section included a copy of the textual description of the bank intrusion from [6], along with the corresponding MUCM (Fig. 5). The third section also comprised three sets of questions, regarding (a) the *background* of the participants, (b) the participants' *understanding of the case*, and (c) the *user acceptance* of the technique. The user acceptance questions were inspired by the Technology Acceptance Model (TAM) [38], reflecting the three TAM-variables *perceived usefulness*, *perceived ease of use* and *intention to use* with 2 items each, giving six items (or questions) in all. At the end of the sheet, open comments were invited. In addition, the respondents were asked how much time they spent on the evaluation sheet and which aids they relied on when answering the questions about understanding of the case (either the *textual description*, the *misuse case map* or *memory*).

The participants spent between 20 and 60 minutes on the task (37 minutes on average). We split the responses in the following four groups, depending on which aids they reported to have relied on when answering the questions about *understanding of the case*. The four groups were TD (9 valid responses relying on the *textual description*), MEM (6 valid responses relying on *memory*), MUCM (6 valid responses relying on the *misuse case map*) and NON-M (4 valid responses *not* using the *misuse case map*). Because most respondents reported relying on more than one aid, the groups overlap considerably. Nevertheless, a comparison of the responses gives a useful first indication of the strengths and weaknesses of MUCM as an aid for understanding a complex intrusion scenario.

Table 1 summarizes the responses according to group. The TD and MUCM groups spent most time on the task, the MEM and NON-M groups the least. With regard to background experience, the groups were quite similar, with average scores between 3.6 and 3.9 on a scale from 1 to 5 (with 5 being highest). The TD and MUCM groups reported slightly lower experience on average than the MEM and NON-M groups, suggesting that less experienced respondents relied more on external aids. This may explain in part why they used more time too. The MUCM group had the highest average percentage (77%) of correct answers to the questions about understanding, whereas the NON-M group had the lowest one (68%). Although time may have played a part, we take this to be an indication that MUCM may indeed be a beneficial aid for understanding complex intrusions. The TD group also had a high average score on understanding, and the MEM group a lower one.

**Table 1.** Responses to the evaluation sheet, grouped according to the aids used when answering the questions about understanding of the case

| Group | Responses | Time (min) | Back-ground | Under-standing | PU | PEOU | ITU |
|---|---|---|---|---|---|---|---|
| **TD** | 9 | 36 | 3.6 | 74% | 3.8 | 2.9 | 3.9 |
| **MEM** | 6 | 30 | 3.9 | 71% | 3.9 | 2.5 | 3.8 |
| **MUCM** | 6 | 37 | 3.7 | 77% | 3.9 | 3.3 | 3.9 |
| **NON-M** | 4 | 30 | 3.8 | 68% | 3.9 | 2.5 | 4 |

On average, the four groups rated the perceived usefulness (PU) of MUCMs similarly, from 3.8 to 3.9 (again on a scale from 1 to 5 with 5 being highest). The average ratings on perceived ease of use (PEOU), however, were considerably higher for MUCM (3.3) than for NON-M (2.5), suggesting that perceived ease of use played an important role in the respondents' decisions whether to use the MUCM or not when answering questions about the case. The TD group also had a high average rating and MEM a low one, on PEOU. In general, the scores on PU were higher than for PEOU, indicating that the somewhat elaborate MUCM notation used in the evaluation was perceived as complex. On intention to use (ITU), however, the average ratings were nearly identical between the groups (from 3.8 to 4.0). Surprisingly, intention to use

misuse case maps in the future was highest for the NON-M group that had not used MUCMs at all. Because the evaluation was preliminary, we do not address validity issues here.

We received written and oral comments on the following issues: the notation was hard to understand although it could become easier to read with time; the map contained too much detail; the map contained too little detail; UML sequence diagrams may be a better alternative; the component concept is unclear because it mixes physical and logical entities; and MUCMs are good for analysis but maybe not for communication. We plan to address these comments in the further development of the technique.

## 6 Conclusions and Future Work

The paper has introduced a new attack modeling technique, *misuse case maps (MUCM)*, that combines an attacker's behavioral view of the proposed software system with an architectural view. The purpose of misuse case maps is to offer a representation technique with the potential to include a wider group of stakeholders, such as domain experts and regular software developers, in security considerations already during the earliest development phases. The technique and its notation was illustrated through a multi-stage bank intrusion described in the literature. Results from a preliminary evaluation were also reported, indicating that MUCM may indeed be a beneficial aid for understanding complex intrusions.

Of course, the preliminary evaluation is severely limited. It used only a small example, which precluded statistical analysis. The evaluation was not controlled, and the subjects were colleagues and other contacts who might have been positively biased towards our proposal. Hence, further empirical evaluations are clearly needed, for example investigating different complex intrusion scenarios for the future like in [39, Chapter 3]. They should involve more subjects working under more controlled conditions.

Future work on MUCMs should address issues such as how to avoid overly complex, spaghetti-like maps, how to best communicate intrusions to domain experts and regular software developers, and how to involve them in the vulnerability exploring and mitigating process. Further evaluations and practical studies will use MUCMs in increasingly realistic settings. We intend to combine MUCM with other attack modelling and security analysis techniques. We also plan to provide practical guidelines to establish a security requirements method and provide tool support for it. The method should perhaps be further extended to consider *anti-functional requirements* in general, addressing *safety requirements* in particular in addition to security. Important questions to address will be when and how to apply the security and safety experts' knowledge and how to manage the different types of information that is generated from the cooperation between customers, domain experts and developers.

**Acknowledgement.** This work was performed as part of the ReqSec project (www.idi.ntnu.no/~guttors/reqsec/) and funded by the Norwegian Research Council.

## References


1. Barnum, S., Sethi, A.: Attack Patterns as a Knowledge Resource for Building Secure Software. In: OMG Software Assurance Workshop (2007)
2. Koziol, J., et al.: The shellcoder's handbook: discovering and exploiting security holes. John Wiley & Sons, Chichester (2004)
3. Hoglund, G., McGraw, G.: Exploiting Software: How to Break Code. Addison-Wesley, Boston (2004)
4. Amyot, D.: Use Case Maps Quick Tutorial (1999), http://www.usecasemaps.org/pub/UCMtutorial/UCMtutorial.pdf
5. Buhr, R., Casselman, R.: Use case maps for object-oriented systems. Prentice-Hall, Inc., Upper Saddle River (1995)
6. Mitnick, K.D., Simon, W.L.: The art of intrusion: the real stories behind the exploits of hackers, intruders & deceivers. Wiley, Chichester (2005)
7. Schneier, B.: Secrets & lies: digital security in a networked world. John Wiley & Sons, Chichester (2000)
8. Amoroso, E.G.: Fundamentals of computer security technology. Prentice-Hall, Inc., Upper Saddle River (1994)
9. Liu, L., Yu, E., Mylopoulos, J.: Security and privacy requirements analysis within a social setting. In: Proc. RE 2003, vol. 3, pp. 151–161 (2003)
10. Lin, L., et al.: Using abuse frames to bound the scope of security problems (2004)
11. McDermott, J., Fox, C.: Using abuse case models for security requirements analysis (1999)
12. Sindre, G., Opdahl, A.L.: Eliciting security requirements with misuse cases. Requirements Engineering 10(1), 34–44 (2005)
13. Firesmith, D.J.: Security use cases. Technology 2(3) (2003)
14. Giorgini, P., et al.: Modeling security requirements through ownership, permission and delegation. In: Proc. of RE, vol. 5, pp. 167–176 (2005)
15. Van Lamsweerde, A., et al.: From system goals to intruder anti-goals: attack generation and resolution for security requirements engineering. In: Requirements Engineering for High Assurance Systems (RHAS 2003), vol. 2003, p. 49 (2003)
16. Dimitrakos, T., et al.: Integrating model-based security risk management into eBusiness systems development: The CORAS approach. In: Monteiro, J.L., Swatman, P.M.C., Tavares, L.V. (eds.) Proc. 2nd Conference on E-Commerce, E-Business, E-Government (I3E 2002), pp. 159–175. Kluwer, Lisbon (2002)
17. Jurjens, J.: UMLsec: Extending UML for secure systems development. In: Jézéquel, J.-M., Hussmann, H., Cook, S. (eds.) UML 2002. LNCS, vol. 2460, pp. 412–425. Springer, Heidelberg (2002)
18. Lodderstedt, T., et al.: SecureUML: A UML-based modeling language for model-driven security. In: Jézéquel, J.-M., Hussmann, H., Cook, S., et al. (eds.) UML 2002. LNCS, vol. 2460, pp. 426–441. Springer, Heidelberg (2002)
19. Rodriguez, A., Fernandez-Medina, E., Piattini, M.: Towards an integration of security requirements into business process modeling. In: Proc. of WOSIS, vol. 5, pp. 287–297 (2005)
20. Rodriguez, A., Fernandez-Medina, E., Piattini, M.: Capturing Security Requirements in Business Processes Through a UML 2.0 Activity Diagrams Profile. In: Roddick, J., Benjamins, V.R., Si-said Cherfi, S., Chiang, R., Claramunt, C., Elmasri, R.A., Grandi, F., Han, H., Hepp, M., Lytras, M.D., Mišić, V.B., Poels, G., Song, I.-Y., Trujillo, J., Vangenot, C. (eds.) ER Workshops 2006. LNCS, vol. 4231, pp. 32–42. Springer, Heidelberg (2006)

21. Schumacher, M., et al.: Security Patterns: Integrating Security and Systems Engineering. Wiley, Chichester (2005)
22. Boswell, A.: Specification and validation of a security policy model. IEEE Transactions on Software Engineering 21(2), 63–68 (1995)
23. Hall, A., Chapman, R.: Correctness by construction: Developing a commercial secure system. IEEE Software, 18–25 (2002)
24. Buhr, R.J.A.: Use case maps for attributing behaviour to system architecture. In: 4th International Workshop of Parallel and Distributed Real-Time Systems (1996)
25. Buhr, R.J.A.: Use case maps as architectural entities for complex systems. IEEE Transactions on Software Engineering 24(12), 1131–1155 (1998)
26. Woodside, M., Petriu, D., Siddiqui, K.: Performance-related completions for software specifications. In: 24th International Conference on Software Engineering (2002)
27. Liu, X., Peyton, L., Kuziemsky, C.: A Requirement Engineering Framework for Electronic Data Sharing of Health Care Data Between Organizations. In: MCETECH (2009)
28. Mussbacher, G., Amyot, D., Weiss, M.: Visualizing Early Aspects with Use Case Maps. In: Rashid, A., Aksit, M. (eds.) Transactions on AOSD III. LNCS, vol. 4620, pp. 105–143. Springer, Heidelberg (2007)
29. Wu, W., Kelly, T.P.: Deriving safety requirements as part of system architecture definition. In: Proceedings of the 24th International System Safety Conference, Albuquerque (2006)
30. Wu, W., Kelly, T.: Managing Architectural Design Decisions for Safety-Critical Software Systems. In: Hofmeister, C., Crnković, I., Reussner, R. (eds.) QoSA 2006. LNCS, vol. 4214, pp. 59–77. Springer, Heidelberg (2006)
31. Alexander, I.: Misuse cases: Use cases with hostile intent. IEEE Software 20(1), 58–66 (2003)
32. Sindre, G.: A look at misuse cases for safety concerns. International Federation for Information Processing Publications - IFIP, vol. 244, p. 252 (2007)
33. Stålhane, T., Sindre, G.: A comparison of two approaches to safety analysis based on use cases. In: Parent, C., Schewe, K.-D., Storey, V.C., Thalheim, B. (eds.) ER 2007. LNCS, vol. 4801, pp. 423–437. Springer, Heidelberg (2007)
34. Stålhane, T., Sindre, G.: Safety Hazard Identification by Misuse Cases: Experimental Comparison of Text and Diagrams. In: Czarnecki, K., Ober, I., Bruel, J.-M., Uhl, A., Völter, M. (eds.) MODELS 2008. LNCS, vol. 5301, pp. 721–735. Springer, Heidelberg (2008)
35. Sindre, G., Opdahl, A.L.: Misuse Cases for Identifying System Dependability Threats. Journal of Information Privacy and Security 4(2), 3–22 (2008)
36. Diallo, M.H., et al.: A comparative evaluation of three approaches to specifying security requirements. In: Proc. REFSQ 2006, Luxembourg (2006)
37. Opdahl, A.L., Sindre, G.: Experimental comparison of attack trees and misuse cases for security threat identification. Information and Software Technology 51(5), 916–932 (2009)
38. Davis, F.D.: Perceived usefulness, perceived ease of use, and user acceptance of information technology. MIS quarterly 13(3), 319–340 (1989)
39. Lindqvist, U., Cheung, S., Valdez, R.: Correlated Attack Modeling, CAM (2003)