TOWARDS A HACKER ATTACK REPRESENTATION METHOD

Peter Karpati, Guttorm Sindre
Dept. of Computer and Information Science, Norwegian University of Science and Technology, Sem Sælands vei 7-9, Trondheim, Norway

Andreas L. Opdahl
Dept. of Information Science and Media Studies, University of Bergen, Bergen, Norway

Keywords: Intrusion analysis, Security requirements, Misuse case, Attack tree, Attack pattern.

Abstract: Security must be addressed at an early stage of information systems development, and one must learn from previous hacker attacks to avoid similar exploits in the future. Many security threats are hard to understand for stakeholders with a less technical background. To address this issue, we present a five-step method that represents hacker intrusions diagrammatically. It lifts specific intrusions to a more general level of modelling and distils them into threats that should be avoided by a new or modified IS design. It allows involving different stakeholder groups in the process, including non-technical people who prefer simple, informal representations. For this purpose, the method combines five different representation techniques that together provide an integrated view of security attacks and system architecture. The method is illustrated with a real intrusion from the literature, and its representation techniques are tied together as a set of extensions of the UML metamodel.

1 INTRODUCTION

Experience tells us that hackers can be very creative, to the point of routinely beating systems with a considerable focus on security (Mitnick & Simon, 2006). An important lesson for secure systems modelling is to model not only the system with its wanted functionality and security precautions, but also combine system models with representations of real or possible attacks to be able to investigate and learn from them. This is the philosophy behind techniques such as attack sequences (see section 4.1), attack trees (Schneier, 1999), attack patterns (Gegick & Williams, 2005) and misuse cases (Sindre & Opdahl, 2005). Each of them have their strengths and weaknesses. Attack sequence descriptions provide nice high level overviews of complex attack sequences, but they are flat and offer little detail. Attack trees and patterns break threats nicely down into AND/OR hierarchies, but they are not concerned with attack sequences or with the legitimate functionality of the system. Misuse case diagrams show threats in relation to the system's functionality and users, but offer little support for attack sequences or for breaking high level threats down into more detailed ones (Opdahl & Sindre, 2009).

Furthermore, none of the established techniques show the relationship between threats and system architecture although architecture is essential for security in several ways: its components suggest typical weaknesses and attack types, and the path each of the system's functions (or use cases) takes through the architecture suggests which weaknesses a user of that function might try to exploit. We therefore recently proposed a new technique providing an integrated view of security attacks and system architecture, namely misuse case maps (Karpati et al., 2010). Compared to the other techniques, misuse case maps have the disadvantage of being more complex, and thus most suitable for technically competent stakeholders.

A single representation technique is not enough to balance the needs of security experts against those of other stakeholders, nor enough for balancing the needs for overview versus detail, behaviour versus structure, function versus architecture etc. This paper therefore introduces the Hacker Attack Representation Method (HARM), an overall method for illustrating complex security attacks using a combination of five techniques. We propose five steps for representing hacker intrusions and offer guidelines for how to use the five techniques together. HARM's key improvement over the current state of practice and research is that it enables many different stakeholders to take part in the discussion of hacker intrusions, which is only available for security experts today. It thus allows knowledge of complex intrusions to be used already in the early analysis stages and it properly links security issues with architecture considerations.

The rest of this paper is structured as follows: Section 2 discusses related work. Section 3 then presents and motivates the overall five-step method. Section 4 illustrates HARM by a running example, elaborating the various techniques included and their purposes. Section 5 outlines how the method can be grounded in the UML metamodel through a small number of extensions, thus enhancing the interoperability between HARM and mainstream modelling approaches. Finally, section 6 discusses the results and concludes the paper.

2 RELATED WORK

Since the start of the new millennium, attention has been drawn to complex intrusions, as opposed to the earlier focus on isolated attack steps. A pioneering effort was the JIGSAW attack specification language (Templeton & Levitt, 2000), which describes the components of an attack in terms of concepts and capabilities. Capabilities are atomic elements defining the circumstances (situation, needed information) required for a particular aspect of an attack to occur. Concepts embody abstract situations that form the attack steps in a complex intrusion. Requirements are defined for concepts and relate capabilities and configurations. If the requirements for a concept are met then the concept holds and can provide new capabilities (meaning that the attack can advance to the next stage). This way, the language allows flexible variations of exploits to create sophisticated attack scenarios. This model can be applied for vulnerability discovery, intrusion detection or attack generation.

JIGSAW was followed by attack graphs (AG) (Sheyner et al., 2002), which drew immediate responses from the security research community. AGs represent all possible attacks on a network. Their nodes and edges express possible actions (usually exploit steps) along with the resulting changes to the state of the network. They are useful for network hardening and penetration testing among other things.

A central merit of JIGSAW and AG was to shift the focus from isolated malicious events to whole intrusion scenarios. This shift led to exploring better methods for secure systems development, more advanced penetration testing tools and more sophisticated intrusion detection systems (IDS). An important aspect of IDS systems is how the alerts are combined to support identification of different intrusion attempts. There are different ways to correlate alerts. For example, (Templeton & Levitt, 2000) suggest to exploit the included definition of requires and provides blocks in the specification to match alerts, but give no specific method for it. (Ning et al., 2002) correlate alerts by matching the consequence of previous alerts and the prerequisite of later ones through hyper-alert correlation graphs based on predicates. It addresses the limitations of JIGSAW by allowing alert aggregation and partial satisfaction of prerequisites. (Cheung et al., 2003) use the components of the EMERALD intrusion detection framework (Neumann & Porras, 1999), which is based on real-time forward-reasoning expert systems.

Recent efforts have proposed to combine security models (from threat modelling through mitigation to testing and inspection into methods for secure software development. The Suraksha project (Maurya et al., 2009) offers a workbench with the possibility to embed security considerations into the system from the earliest stages of software development. The approach suggests the following steps: 1) identifying system objectives, assets and their risks; 2) analyzing functional requirements using UML and developing use cases (UC); 3) applying the STRIDE concept for each UC and developing lightweight misuse case (MUC) diagrams; 4) developing attack trees (AT) from each abstract threat node in the MUC diagrams; 5) using DREAD for each AT; 6) selecting relevant threats; 7) modifying the MUCs and extending them with more details; 8) finalizing the security requirements, considering them as functional requirements, converting the MUCs to UCs and suggesting mitigations in the form of security use cases (SUC); 9) finding appropriate security patterns (SP) and going back to step 1 as long as there are remaining SUCs without an SP. The Suraksha tool supports these steps and we used it to create the MUCs and ATs for this paper.

The SHIELDS project (Tøndel et al., 2010) also proposes to combine MUCs (modelling abstract threats) with ATs (detailing those threats) and to provide references in the MUC diagrams to security activity descriptions that mitigate the threats. A security repository is introduced for storing deliberate models and relations for reuse. The proposal differs from Suraksha by referencing the threats more explicitly and focusing more on reusability of models.

The PWSSec (Process for Web Service Security) approach (Gutierrez et al., 2006) guides developers in the integration of security into the development stages of Web Services (WS)-based software. The process is iterative and incremental, and consists of three stages: 1) the specification of WS-specific security requirements, 2) the definition of the WS-based security architecture and 3) the identification, integration and deployment of WS security standards. Stage 1 produces attack trees (AT) and misuse cases (MUC) as outputs among other models. The leafs of the ATs show the threats which are refined by a set of attack scenarios defined by MUCs. The process also provides a security architecture formed by a set of coordinated security mechanisms (Gutierrez et al., 2005a) in stage 2. The base for this is the WS-based security reference architecture (Gutierrez et al., 2005b) which guides the system designers in the task of allocating the security requirements into the security architecture. The core of the security reference architecture is the WS Security Kernel managing a set of Abstract Security Services thus covering a set of security requirements.

None of these techniques relate threats, attacks, vulnerabilities and mitigations to systems architecture. (Only PWSSec considers architecture and relates security mechanisms to it through the WS-based reference architecture in form of Abstract Security Services.) Use case maps (UCM) outline the architecture of systems and show their relations to specific use cases (of functions), thus providing high-level overviews that support design and development (Buhr & Casselman, 1995; Buhr, 1996). UCMs can include multiple scenarios to represent a chosen aspect of the system's behaviour. The basic UCM notation consists of three types of building stones: 1) runtime components as rectangular boxes, 2) responsibilities bounded to components as crosses, and 3) scenario paths capturing a causal sequence of responsibilities (lines cutting through the responsibilities). A scenario path starts with a filled circle representing pre-conditions or triggering causes and ends in a bar representing post-condition or resulting effect. (See the first three elements in Fig. 2.) We have recently adopted UCMs to deal with security (Karpati et al., 2010). Misuse case maps (MUCM) provide integrated overviews of misuse cases (as exploit paths, the "negative" variant of scenario paths) and system architecture, highlighting vulnerabilities and suggesting possible mitigation points.

3 OUTLINE OF HARM

This section presents an outline of the HARM method. We will first present the method steps, then discuss the relations between the representation techniques and give guidelines for using them together. The following section will present an example of a real attack from the literature.

3.1 Relation to Existing Approaches

HARM retains the focus in Suraksha (Maurya et al., 2009) and SHIELDS (Tøndel et al., 2010) on whole intrusion scenarios. But the two approaches are not directly suitable for our purpose because of the formalisms involved and their limited expressiveness. Compared to attack graphs, we also intend to scale up from a subset of technical attacks (like network attacks) to general attacks, e.g., combinations of social engineering, physical entry and computer hacks. Systems like (Ning et al., 2002; Cheung et al., 2003; Neumann & Porras, 1999) are interesting for us because they are based on complex attack models. For example, (Cheung et al., 2003) utilize attack trees and attack patterns too, but combine them in a different way than HARM does since their aim is a bit different (develop models for multistep attack scenarios generally) and the definition of the attack patterns also differ. Compared to Suraksha and SHIELDS, our approach presents complex intrusions at a detailed level in addition to threats and it focuses less on mitigations. Relative to misuse case maps (Karpati et al., 2010), HARM retains the focus on architecture, but introduces additional perspectives.

PWSSec is a process specifically designed for web services whereas HARM is a general method. PWSSec uses business and security goals as well as organizational security policy to derive attack scenarios (Gutierrez et al., 2005a) while HARM works in an exploratory way starting by intrusion cases . Thus, the two approaches complement each other since the first provides a general design while the second opens up space for creativity and presents the view of always developing, ingenious attackers.

3.2 Method Steps

HARM consists of the following five steps:

1. *Outlining the intrusion*: Make a simple, structured description of the case as alternating attacking activities and outcomes. The case may be the trace of a known multi-stage attack, the plan for a penetration test or the detailed reconstruction of an intrusion at hand. This step may involve the cooperation of security experts, system administrators or system designers. It uses the ASD representation technique.
2. *Detailing the scenarios*: Analyze each attacking activity in further detail in relation to other activities and the architectural context, including the specific vulnerabilities that have been exploited and possible mitigations. This step may involve the same actors as step 1 and it uses MUCMs.
3. *Providing functional context*: Distill essential facts from the detailed scenarios, by introducing a functional and user perspective on the attacking activities, vulnerabilities and mitigations, so that threats and solutions can be considered from different viewpoints (e.g., business, usage, technical viewpoints, etc.). This step uses the MUC diagram representation to encourage involvement of the broadest possible set of stakeholders.
4. *Refining the attack structure*: Relate specific attacks to other attack types in a hierarchy, allowing elicitation of threats from specific vulnerabilities and preparing to lift the appropriate mitigations to a more general level. System designers and software developers might be responsible for finalizing this step, using the AT technique. This step interlaces with step 3 since there can be a lot of mutual influence while creating the MUC diagram and the AT.
5. *Distilling the threats*: Consider in detail the threats and mitigations captured in previous steps to make a design out of it based on expert knowledge embedded in APs. This step involves system/software designers and software developers and security experts if available.

Hence, we adopted the combination of MUCs and ATs from Suraksha and SHIELDS and added further representation techniques. Requirements are elicited an exploratory way: from specific cases towards general designs. Starting with vulnerabilities, exploits and mitigations specified in a limited context (ASD & MUCM), the method helps to generalize them using previously collected knowledge and relations to similar entities (MUC & AT & AP). It also supports further development into well-considered security strategies.

3.3 The Modelling Techniques and their Relations

The method uses the following representation techniques:

- *Attack Sequence Descriptions (ASD)* summarizing attacks in natural language.
- *Misuse Case Maps (MUCM)* depicting the system architecture targeted by the attack and visualizing traces of the exploit(s).
- *Misuse Case (MUC)* diagrams showing threats in relation to wanted functionality.
- *Attack Trees (AT)* presenting the hierarchical relation between attacks.
- *Attack Patterns (AP)* describing an attack in detail with additional information of context and solutions.

Fig. 1 shows the relations between the techniques. ASD is the starting point and contains the information required to create or identify further models. It is specific to the case in focus, just like the MUCM. Beside depicting the architecture and the trace of the intrusion, the MUCM also facilitates discussion about alternative vulnerabilities and mitigations, still oriented by the specific case. The MUC diagram takes a step up and looks at the case from a more general viewpoint, showing the use cases appearing in the ASD and related ones within their functional context. Details unnecessary for threats and mitigation modelling are eliminated. While MUCM and MUC include regular use cases, ATs focus only on the attacks, exploring refinements and alternatives, as well as whether attacks may fit together with other (technical) attacks. They complement MUCs since they lead the focus towards standard textbook threats while MUCs lead it towards problem-specific threats (Opdahl & Sindre, 2009). APs define more details about a type of an attack considering them in their context of prerequisites and other acquired pieces of knowledge.

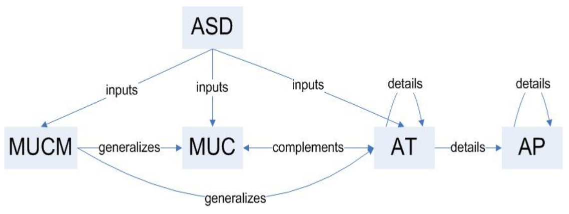

Figure 1: Relations of the applied attack modelling techniques.

4 USING HARM IN PRACTICE

This section illustrates HARM by applying it to a real penetration test described in (Mitnick & Simon, 2006). We present the models introduced at each step and discuss their relationships to one another.

4.1 Outlining the Intrusion: Attack Sequence Description (ASD)

An ASD consists of ordered steps where a step defines an activity of the attacker or relevant outcomes/revelations for the following steps (indicated by ”activity => outcome”), all formulated in natural language. Some remarks can be added to the steps in brackets to help understanding. An ASD is presented from the intruder’s point of view.

The example ASD summarizes an entry into the computer system of a company for penetration test purposes (Mitnick & Simon, 2006, ch. 6). The description in the book does not reveal all the details to prevent the reproduction of the intrusion, thus we lack some information in our models, too.

1. Checking the web server => it runs Apache
2. Checking the firewall => found a hidden default configuration setting allowing in packets with a source UDP or TCP port of 53 to almost all the high ports (above 1023)
3. Trying to mount off the file sys. using NFS => Firewall blocked access to NFS daemon and common system services
4. Using an undocumented Solaris feature (portmapper - rpcbind - bound to port 32770) to get the dynamic port of the mount daemon (mountd) from the portmapper and direct an NFS request to it => the target system's file system was remotely mounted an downloaded
5. Recognizing a PHF vulnerability => trick the PHF CGI script to execute arbitrary commands (by passing the Unicode string for a newline character followed by the shell command to run; PHF is used to access web-based phone book by querying a DB)

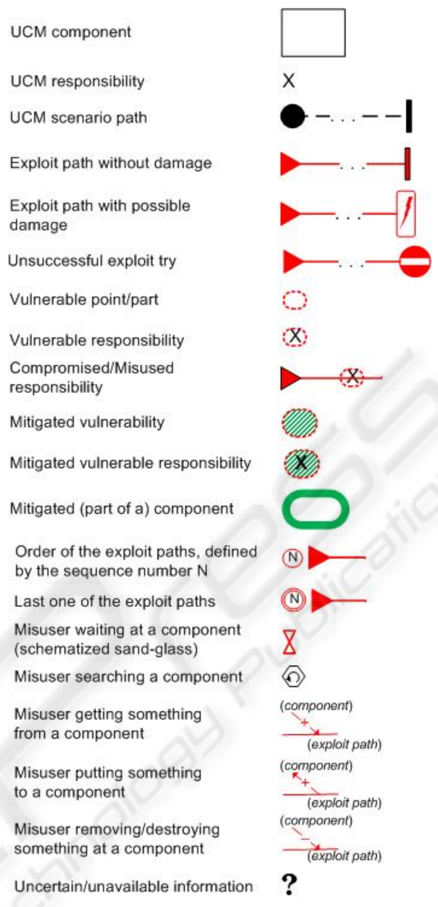
UCM component
UCM responsibility
X
UCM scenario path
Exploit path without damage
Exploit path with possible damage
Unsuccessful exploit try
Vulnerable point/part
Vulnerable responsibility
Compromised/Misused responsibility
Mitigated vulnerability
Mitigated vulnerable responsibility
Mitigated (part of a) component
Order of the exploit paths, defined by the sequence number N
N
Last one of the exploit paths
N
Misuser waiting at a component (schematized sand-glass)
Misuser searching a component
Misuser getting something from a component
(component)
(exploit path)
Misuser putting something to a component
(component)
(exploit path)
Misuser removing/destroying something at a component
(component)
(exploit path)
Uncertain/unavailable information
?

Figure 2: Basic UCM and MUCM notation.

6. Through PHF: found that the Apache server process was running under the “nobody” account (i.e., the computer system was secured) and its configuration file was owned by “nobody” (means: it could be overwritten through the PHF CGI script which was running also under “nobody”) => change httpd.conf so that the server restarts as “root” and wait => the server restarted because of a blackout
7. Installing a backdoor to prevent getting shut out of the system
8. Setting up a sniffer on all e-mail going in and out and searching the (Oracle) DB for CIO’s salary (as a proof for their customer)
9. Continue to penetrate the entire network
10. *Sometime after step 6*: Installing sniffer programs everywhere: a) at the firewall so they were aware of all maintenance work there, b) at a router where a system administrator failed to enter the right password for more times => gaining many administrator passwords to different internal systems

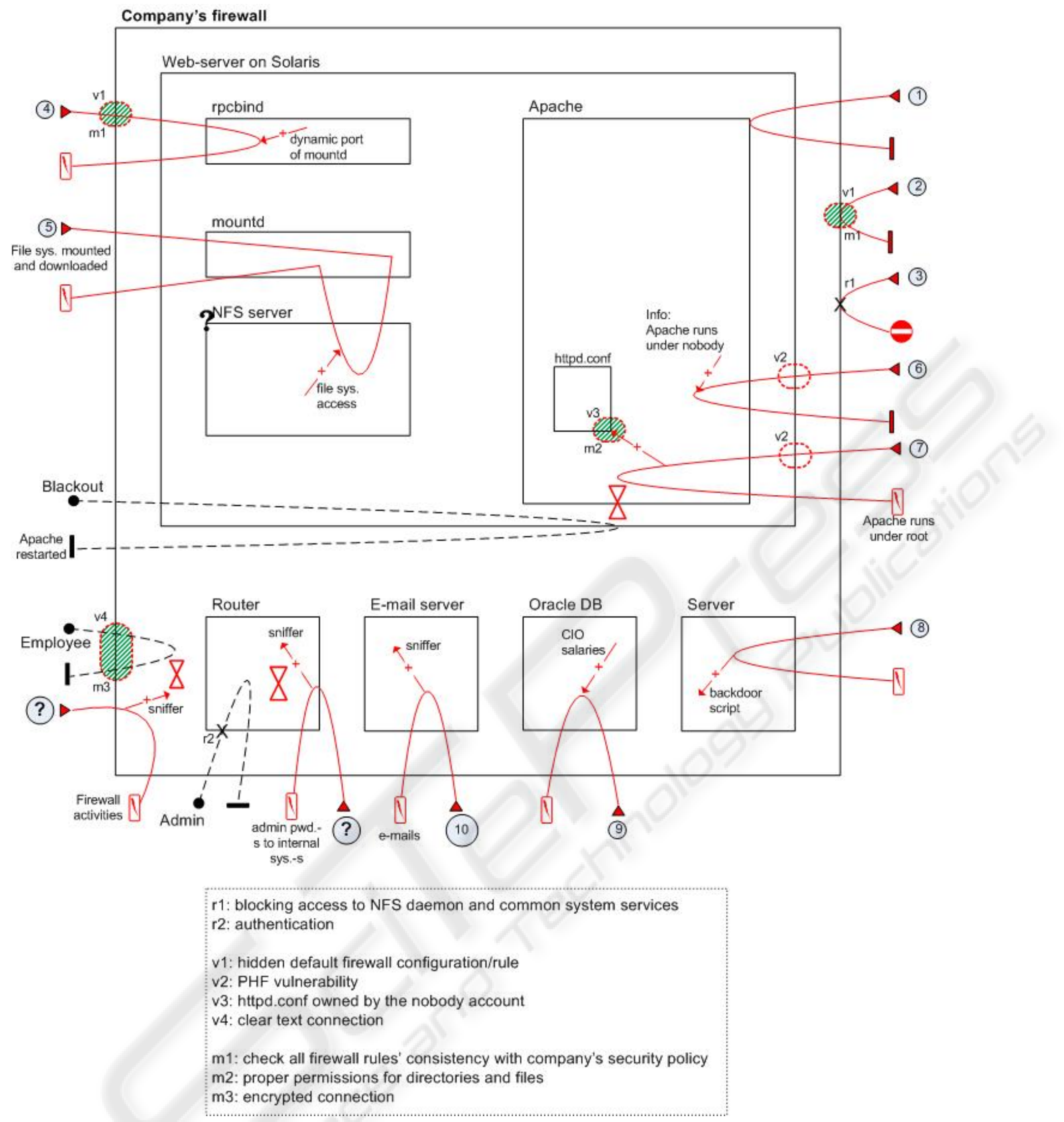
Company's firewall
Web-server on Solaris
rpcbind
dynamic port of mountd
mountd
NFS server
file sys. access
Apache
Info: Apache runs under nobody
httpd.conf
v1
m1
v2
v3
m2
r1
File sys. mounted and downloaded
Blackout
Apache restarted
Apache runs under root
Router
sniffer
E-mail server
sniffer
Oracle DB
CIO salaries
Server
backdoor script
Employee
v4
m3
sniffer
r2
Firewall activities
Admin
admin pwd.-s to internal sys.-s
e-mails
1
2
3
4
5
6
7
8
9
10
r1: blocking access to NFS daemon and common system services
r2: authentication
v1: hidden default firewall configuration/rule
v2: PHF vulnerability
v3: httpd.conf owned by the nobody account
v4: clear text connection
m1: check all firewall rules' consistency with company's security policy
m2: proper permissions for directories and files
m3: encrypted connection

Figure 3: MUCM based on the penetration test ASD (steps numbers are different).

Next, the MUCM is built according this description.

4.2 Detailing the Scenarios: Misuse Case Maps (MUCM)

Misuse case maps (MUCMs) (Karpati et al., 2010) combine misuse cases (MUC) (Sindre & Opdahl, 2005) and use case maps (UCM) (Buhr & Casselman, 1995), presenting an integrated view of security issues and system architecture. MUCMs address security requirements by focusing on vulnerabilities, threats and intrusions (inherited from MUCs) from an architectural point of view (inherited from UCMs). Their notation is based on the UCM notation and extended by vulnerabilities, exploits and mitigations.

The system may have vulnerable points (such as authentication responsibility) or parts (such as components without up-to-date security patches) which are suspect to threats. Mitigations can help to counter the threats and appear in the MUCM as desired possibilities which translate to security requirements later. Misuses are depicted by the exploit path's crossing of a vulnerable point or part. The notation offers many further possibilities that we will not describe in detail here, such as labels attached to other symbols and a question-mark notation for still unclear steps.

The example MUCM shown in Fig. 3 is based

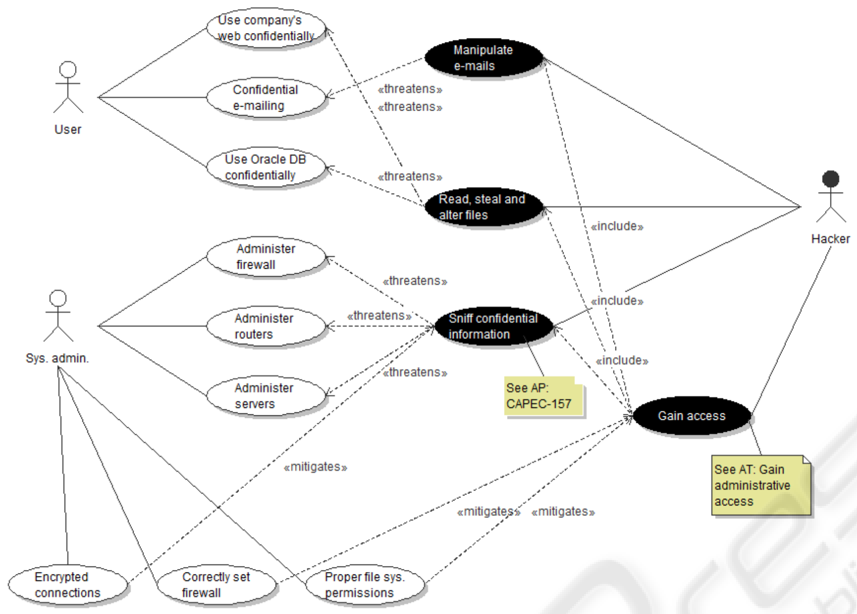

Figure 4: MUC generalized from the penetration test example.

on the ASD but the numberings of the steps are different. We incorporated mitigations as well but only those which were mentioned in connection with the case. The list of the specific vulnerabilities, addressed responsibilities and mitigation possibilities can be found below the map. The case will be generalized in a MUC and an AT.

4.3 Providing Functional Context: Misuse Cases (MUC)

Misuse cases (MUC) (Sindre & Opdahl, 2005) have become popular for security requirements elicitation and threat modelling. They complement use cases (UC) for security purposes by extending them with *misusers*, *misuse cases* and *mitigation use cases*, as well as new relations like *threatens* and *mitigates*. MUC diagrams use an inverted notation and are combined with regular UCs. MUCs facilitate discussion among stakeholders including regular developers with no special security training. Fig. 4 shows how a MUC diagram deals with non-functional security issues by representing the view of an attacker (Alexander, 2003). We allow further references to ATs and APs in a MUC in order to show the relations among them.

Fig. 4 shows how the use and misuse of the system is lifted to a more general level. The vulnerabilities take the form of threats or misuses and links to other models appear which elaborate more on them. The mitigations also appear in a more general context: the threat of sniffing confidential information anywhere in the system is mitigated by encrypted connection in the MUC while the vulnerability of clear text connections at one place in the system (firewall maintenance) was mitigated by the encrypted connections at that specific place in the MUCM. The customers and field experts can also easily input their knowledge and desires here, for example discussing what is confidential and how much cost is it worth to secure this confidentiality.

4.4 Refining the Attack Structure: Attack Trees (AT)

Attack trees (AT) ("threat trees" by Microsoft) provide a structured way for describing a high level attack and various ways for its realization (Schneier, 1999). The high level attack is in the root node of the tree and is decomposed AND/OR into lower-level attacks that must succeed to realize the higher-level ones. Nodes in the AT can have values and thus can answer questions like "Which is the cheapest attack?" or "Which is the best low-risk, low-skill attack?". ATs have diagrammatical and textual outline notation (Schneier, 2000). They can be used to evaluate proposed designs but are also applicable at an early requirements stage. We use them to detail general aspects of the intrusion together with attack patterns like in Fig. 5 for the penetration test case (see the labels).

The hacker in the example case acquired a first access to the system by using the PHF vulnerability and thus bypassing authentication. Fig. 5 shows another alternative for getting access: through regular authentication by finding (out) identifiers and passwords. There are many alternatives to achieve this which are depicted in a separate AT not shown here. It shows how we can gain more general threats from a specific vulnerability.

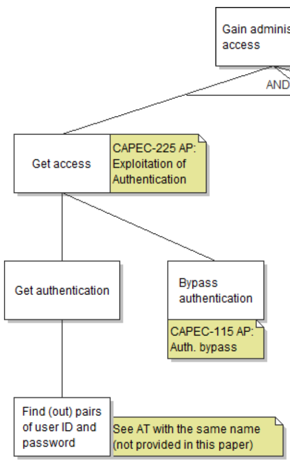
Gain administrative access
AND
Get access
CAPEC-225 AP: Exploitation of Authentication
Install hacker tools
Capture data traffic
Get authentication
Bypass authentication
CAPEC-115 AP: Auth. bypass
Turn off antivirus sw.
Find (out) pairs of user ID and password
See AT with the same name (not provided in this paper)

Figure 5: AT for the MUC from the penetration test example.

4.5 Distilling the Threats: Attack Patterns (AP)

An attack pattern (AP) describes the approach used by attackers to generate an exploit against software. It consists of a minimal set of nodes in an attack tree that achieves the goal at the root node (Barnum, 2007). An AP may be a subtree of an AT from the root node to at least one leaf node. In a simple case, when the AT has only OR branches, it is a path from a leaf node to the root and called attack path. APs detail ATs which provide a higher level view.

APs are described with the following information: *Pattern name and classification, Attack prerequisites, Description, Related vulnerabilities or weaknesses, Method of attack, Attack motivation-consequences, Attacker skill or knowledge required, Resources required Solutions and mitigations, Context description, References* (Barnum, 2007). An extensive collection of AP-s can be found at capec.mitre.org.

The APs referenced in figure 4 and 5 give further details on the threats in focus. It may include alternatives for the solutions already acquired in the previous steps of HARM thus adding new mitigation possibilities. Since APs are created by security experts, they also enhance the quality of the model. Although we used only one intrusion scenario here, more can be modelled together in a practical situation, resulting more other diagrams.

5 TOWARDS A METAMODEL FOR HARM

To provide a stronger backbone for HARM, and to prepare for tool support and more formal analyses, this section will outline a metamodel for HARM as a possible extension of the UML metamodel, starting with its two most central behavioural concepts; Behaviour and Action (OMG, 2009). A Behaviour belongs to a Classifier (which thus becomes a BehaviouralClassifier) and is intended to represent *complex* behaviours. An Action can be part of a Behaviour and is intended to represent *atomic* behaviours. Both Actions and Behaviours can be executed. The other BehaviouralClassifier besides Behavior in UML is Actor.

To these concepts, the HARM metamodel adds Misbehaviours, MisbehaviouredClassifiers, AntiActions and Misusers. Each of them is a "negative variant" of the corresponding UML concept. For the moment, we define them as specialisations of their respective UML metaclasses. Subclassing is appropriate from an abstract syntactical perspective, because the new concepts can enter the same relationships as the original concepts (although they sometimes take an inverted meaning). Subclassing is also appropriate from a concrete syntactical perspective, as the new concepts will be drawn with the same icons and connection points as the original concepts (although they will usually be shaded or filled). But subclassing is not quite correct from a semantic perspective, because the new concepts have modalities that are different from the original ones. For example, an Action represents (atomic) behaviour that is both permitted and wanted, whereas an AntiAction represents behaviour that is unwanted and should be forbidden. Apart from modality, though, the semantics of the new and original concepts remain the same. It is easy to provide a cleaner semantics by introducing a new common superclass, e.g., "BasicAction", to capture the common syntax and semantics of Actions and AntiActions, which now become siblings, but we leave this for further work as it would interfere with the existing UML definition.

To incorporate use case maps into the HARM metamodel, we take as our starting point the proposal of (Amyot & Mussbacher, 2000) of basing the definition of use case maps on UML's activity diagrams. We retain this strategy, although activity diagrams have since moved from state-transition to place-transition based semantics. Specifically, a path in a use case map is considered a type of UML Activity (a subclass of Behaviour), and the path elements in the use case maps are considered ActivityNodes. Start and end points in the map correspond to Initial- and FinalNodes, whereas AND and OR elements correspond to ControlNodes. The various action elements in the map correspond to particular subtypes of UML Actions.

To incorporate misuse cases into the HARM metamodel, we introduce MisuseCase as an additional MisbehaviouredClassifier in addition to Misuser. MisuserCase and Misuser inherit all the regular use-case relationships, to which we add two new DirectedRelationships: Threatens and Mitigates.

To incorporate misuse case maps into the HARM metamodel, we take the incorporation of use case maps above as our starting point. A misuse case map is just a use case map where at least one path (a Behaviour) has been turned into an exploit path (a Misbehaviour), thus comprising at least one AntiAction. Like Aymot and Mussbacher, we have to leave for further work the relation between misuse (and use) case maps and the architectural concepts from UML's other language units, such as its deployment diagrams.

To incorporate attack trees and patterns, we can use the MisuseCase concept again, or add a new Attack subclass of MisbehaviouredClassifier. As pointed out in (Sindre et al., 2002), generalisation relationships, already inherited from UML's Classifier concept, can be used to account for OR-decomposition of attacks, whereas include relationships can account for AND-decomposition.

Due to limited space, we leave attack sequence descriptions for further work.

6 CONCLUSIONS AND FURTHER WORK

We have presented the Hacker Attack Representation Method (HARM), which illustrates complex security attacks through a combination of five representation techniques. We have also offered guidelines for how to use the techniques together, and outlined how HARM can be defined as an extension to the UML metamodel. The key improvement over state of the art is that it enables different groups of stakeholders to understand and take part in discussions of hacker intrusions. It allows knowledge of complex intrusions to be used already in the early analysis stages and it links security considerations with architecture.

Our running example based on a real penetration test indicates that HARM may indeed be a useful contribution to the arsenal of available IS and SE methods. Its combination of techniques allows the adjustment of representations to the stakeholders' different backgrounds. Each group of stakeholders can work with appropriate diagrams at an appropriate abstraction level. When customers and domain experts are involved in the discussion, MUC and MUCM might be the best choice, whereas AT can be a great addition for requirements engineers and developers. APs can help to transfer knowledge between regular developers and security experts.

We admit that proposing yet another modelling method for information systems and software engineering must always be done with care, given that the number of available methods is already high and rising. However, as observed already decades ago, one single method cannot cover the wide range of different system tasks (Benyon & Skidmore, 1987). Also, when several different diagram types are used to illustrate a complex system, each diagram can be kept simpler by focusing on particular aspects of the system under discussion. There is a continual need for developing improved approaches, and the diversity of modelling techniques is therefore not only inevitable, but perhaps even desirable (Steele & Zaslavsky, 1993). In particular, there is a need to develop better methods for system tasks that have previously not been given enough attention, e.g.: security in early-stage system analysis (Mead & Stehney, 2005).

The metamodel outlined suggests that UML may be a good starting point for defining HARM in more detail. Despite its known weaknesses, in particular regarding semantics, UML's metamodel is well suited because it is widely accepted and incorporates use cases, which is the "positive variant" of one of the central techniques in HARM. UML also defines several of the other concepts needed by HARM. Defining it as an extension of the UML metamodel also prepares for including "negative variants" of further representation techniques, such as the mal-activity diagrams proposed in (Sindre, 2007). A negative consequence of using the UML metamodel is that our proposal inherits several of UML's weaknesses. For example, Actions and Behaviours become separated, with neither being a subclass of the other, and they are also distinct from BehaviouredClassifiers. In consequence, the relationships between paths and path elements in UCM/MUCM, misuse cases in MUC, primitive attacks in AT/AP etc. often become indirect even when the represented phenomena appear similar.

Our work on HARM so far has focussed on capturing technical intrusions. In the future we also plan to investigate other types of intrusion, such as physical ones and social engineering attacks. Generally we will explore further how the HARM techniques with vulnerability taxonomies can be used for attack and test generation. Further work is needed to add detail, e.g., about how previous attacks are selected and system boundaries defined, how multiple misuse case maps are distilled into misuse case diagrams, how requirements can be derived from attack patterns etc. As a consequence, it is possible that the detailed five-step method will be elaborated and reorganised, although the broad progress of HARM will most likely remain.

ACKNOWLEDGEMENTS

This work was done in the ReqSec project funded by the Norwegian Research Council.

REFERENCES

Amyot, D., Mussbacher, G. (2000) On the Extension of UML with Use Case Maps Concepts. *Proc. UML 2000*, pp 16-31.

Alexander I. (2003) Misuse Cases: Use Cases with Hostile Intent, *IEEE Software*, 20(1):58-66.

Barnum, S. (2007) Attack Patterns as a Knowledge Resource for Building Secure Software, In *A. Sethi* (ed.) Cigital: OMG Software Assurance WS

Benyon, D., Skidmore, S. (1987) Towards a Tool Kit For the Systems Analyst, The Computer Journal 30(1):2-7

Buhr R. J. A. (1996) Use case maps for attributing behaviour to system architecture, *Proc. 4th Int. WS on Parallel and Distributed Real-Time Systems*, p.3

Buhr R.J.A., Casselman R.S. (1995) Use Case Maps for Object-Oriented Systems, Prentice Hall

Cheung, S., Lindqvist, U., Valdez, R. (2003) Correlated Attack Modeling (CAM), Final Technical Report by SRI International, October 2003

Gegick, M., Williams, L., (2005) Matching attack patterns to security vulnerabilities in software-intensive system designs, *Proc. SESS05* — building trustworthy applications, pp 1-7

Gutierrez, C., Fernandez-Medina, E., Piattini, M. (2005a), Web services enterprise security architecture: a case study. *Proc. WS on Secure Web Services* (SWS'05) , Fairfax, VA, USA.

Gutierrez, C., Fernandez-Medina, E., Piattini, M. (2005b), Towards a Process for Web Services Security, *Proc.WOSIS'05 at ICEIS'05*, Miami, Florida, USA.

Gutierrez, C., Fernandez-Medina, E., Piattini, M. (2006), PWSSec: Process for Web Services Security, In *Proc. ICWS '06*, pp.213-222, 18-22

Karpati P., Sindre G., Opdahl A. L. (2010) Illustrating Cyber Attacks with Misuse Case Maps, *Proc. REFSQ*

Maurya, S., Jangam, E., Talukder, M., Pais, A.R. (2009) Suraksha: A security designers' workbench. *Proc. Hack.in 2009*, pp. 59–66.

Mead, N.R, Stehney, T. (2005) Security Quality Requirements Engineering (SQUARE) Methodology. In *Proc SESS'05*. St. Louis, MO, May 15-16, 2005

Mitnick K. D., Simon W. L. (2006) The Art of Intrusion, Wiley Publishing Inc.

Neumann, P.G., Porras, P.A.. (1999) Experience with EMERALD to date. *Proc WS on Intrusion Detection and Network Monitoring*, pp:73-80

Ning, P., Cui, Y., Reeves, D.S. (2002) Constructing attack scenarios through correlation of intrusion alerts. *Proc. 9th ACM conf. on CCS*, pp: 245-254

OMG Unified Modeling LanguageTM (OMG UML), Superstructure Version 2.2, Feb. 2009

Opdahl A. L., Sindre, G. (2009) Experimental Comparison of Attack Trees and Misuse Cases for Security Threat Identification, Information and Software Technology, 51(5):916-932

Schneier, B. (1999) Attack Trees, Dr. Dobb's Journal

Schneier, B. (2000) Secrets and Lies: Digital Security in a Networked World, Wiley.

Sheyner, O., Haines, J., Jha, S., Lippmann, R., Wing, J.M. (2002) Automated Generation and Analysis of Attack Graphs, *Proc. IEEE Symposium on Security and Privacy*, p.273, May 12-15

Sindre, G. (2007). Mal-Activity Diagrams for Capturing Attacks on Business Processes. Lecture Notes in Computer Science, vol. 4542. pp 355-366

Sindre, G., Opdahl A.L. (2005). Eliciting Security Requirements with Misuse Cases. Requirements Engineering 10(1): 34-44

Sindre, G., Opdahl, A.L., Brevik, G.F. (2002) Generalization/Specialization as a Structuring Mechanism for Misuse Cases. *Proc. SREIS'2002*.

Steele, P., Zaslavsky, A. (1993) The Role of Metamodels in Federating System Modelling Techniques, In *Proc ER'93*, Dallas, USA, pp 301-12

Templeton, S.J., Levitt, K. (2000) A requires/provides model for computer attacks, *Proc. WS on New security paradigms*, pp.31-38.

The Mitre Corp. (2010): Common Attack Pattern Enumeration and Classification, capec.mitre.org. Accessed: 30.3.2010.OMG (2009)

Tøndel, I.A., Jensen, J., Røstad, L. (2010) Combining misuse cases with attack trees and security activity models *Proc. OSA workshop*.