Genie in a Model? Why Model Driven Security will not secure your Web Application

Christoph Hochreiner[1], Peter Frühwirt[1]*, Zhendong Ma[2], Peter Kieseberg[1],
Sebastian Schrittwieser[3], and Edgar Weippl[1]
[1]*SBA Research, Austria*
{chochreiner, pfruehwirt, pkieseberg, eweippl}@sba-research.org
[2]*Austrian Institute of Technology, Austria*
zhendong.ma@ait.ac.at
[3]*St. Pölten University of Applied Sciences, Austria*
sebastian.schrittwieser@fhstp.ac.at

Abstract

More often a new software development methodology called Model Driven Engineering (MDE) is used to increase productivity by supporting powerful code generation tools, which allows a less error-prone implementation process. However the idea of modeling system aspects during the design phase - so called Model Driven Security (MDS) - was proposed by the scientific community decades ago and yet it is still unclear whether MDS can improve the security of a software project. In this paper we provide a comprehensive evaluation of current MDS approaches based on a web application scenario in regards to the most common web security attacks. We discuss their strengths and limitations as well as the practicability of MDS for modern web application security in general.

Keywords: model engineering, model driven security, security engineering

1 Introduction

Within the course of the past few years, software correctness and secure software became more and more important. One of the most effective approaches to obtain software correctness is to make use of Model Driven Engineering (MDE). Due to the widespread diffusion of modeling languages, like the Uniform Modeling Language (UML), several developers and research groups picked up the development of MDE techniques and proposed sophisticated tools for code generation. These techniques and tools can be used, to reduce design flaws as well as bugs, which are often introduced during the different phases in the software development process. Flaws are typically introduced in the early stages of the planning phase. These flaws can lead to conceptual problems later in the implementation phase. Modeling techniques are designed to support the software architect during the planning phase to detect and resolve well-known design flaws to obtain a solid software architecture. The second type of software defects are bugs. These defects are introduced during the development phase by software developers. The application of MDE in this phase can eliminate these bugs with the application of automatic code generation approaches. Assumed that the software models are flawless, the abstract model can be automatically translated into bug-free code. Apart from the software development process, techniques like model-based testing, model checking and model validation can be used to verify the reliability a program implementation in reference to its model.

Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications, volume: 5, number: 3, pp. 44-62
*Corresponding author: Favoritenstrasse 16, 1040 Wien, Austria; Tel: +43-699-17941418, Web: http://www.sba-research.org

Due to the tremendous success of MDE, the scientific community [1, 2] suggested to apply these approaches to the security domain to improve the software quality, especially in respect to security. Model Driven Security (MDS) techniques can be applied to the same software development phases, which were already mentioned above. Design-based vulnerabilities can be eliminated in the requirements engineering and the design phase by applying goal oriented system analysis approaches and model checking techniques. In the implementation phase, automatic code generation tools can eliminate bugs, which are typically introduced by software developers. Although the MDE and MDS techniques seem very similar, there are several important aspects introduced by the MDS techniques.

This paper provides an overview about the major approaches and compares these approaches based on the example of a simple web application. The goal is to evaluate, how these techniques can support a software architect to incorporate important security concepts and detect potential security flaws already in the design phase. Due to the non-functional nature of security aspects, MDS is heavily influences by the open world assumption in contrast to MDE. For MDE it is vital to implement all aspects of the model, while for MDS, some parts of the implementation can be omitted, without affecting the functional aspects of the program. Due to the open world assumption it is infeasible to model all possible attacks. Based on the results of our evaluation, we believe, that MDS provides excellent tools to design secure programs, but it lacks the ability to actually enforce these security aspects.

The content of this paper is based on our contribution to the latest AsiaARES-conference [3][1], where we outlined the basic methods. The main contributions of this paper can be defined as follows: We show which types of common threats in web application scenarios can be modeled and to what degree the corresponding security measures are enforced by the different modeling techniques. Furthermore, we provide the analysis of our experimental assessment of current security modeling techniques based on a typical web application scenario. Additionally, we discuss the practicability of MDS for the secure development of web applications. In contrast to the original paper we have updated our findings to the OWASP Top 10 of 2013, provide extended details of the different modeling methods and propose a strategy for how to apply and combine the modeling methods considering their characteristics in Section 6.

2 Related work

Models enable us to use simplified representations and levels of abstraction to solve complex real-world problems. In particular, graphical models are very helpful for understanding and communicating problems in an intuitive way. MDS exploits these advantages by tightly integrating models into the process of design, analysis and implementation of secure software systems. Consequently, existing work on MDS mainly deals with three issues: modeling, model-based security analysis, and model transformation [4]. Modeling focuses on how to capture and model a system and how to model the system design along with security requirements. Due to the de facto standard of UML in software engineering, most approaches extend UML to integrate security aspects. For example, UMLsec [5, 1] defines a UML profile to include security requirements such as confidentiality and secure information flow in the UML diagram. SecureUML [6, 7] adds syntax and semantics for modeling system design with access control requirements to UML. The syntax is defined by using the OMG Meta-Object Facility (MOF) standard for meta models and UML profiles.

Modeling is usually the first step in the MDS process. The modeled systems are used for reasoning about their security properties in the course of the security analysis. To this end, a large amount of related work exists, which also includes traditional formal methods like model checking and theorem proving that uses formal languages to specify (hence model) and verify the security of a system.

[1] This paper is an extended version of the paper [3], presented at the Information Communication Technology-EurAsia Conference 2014 (ICT-EurAsia 2014), Bali, Indonesia, April 14–17, 2014.

In [8], the authors provided a taxonomy evaluation of different state-of-the-art approaches for model driven engineering. The taxonomy was proposed purely theoretically, still, to the best of our knowledge, there has been no structured practical comparison of the actual techniques with respect to implementing a real-life scenario. Our work is focused towards the practical applicability and effectiveness of model driven engineering approaches such as Lloyd and Juerjens [9] did when they applied the UMLsec and JML (Java Modelling Language) approaches to practically evaluate a biometric authentication system. They used UMLsec to emphasis and specify security requirements for modeling threat levels and to generate a JML from the UMLsec model. Gruenbauer et al. [10] evaluated layered security protocols which are used to develop a bank application that contains personal sensitive data. Their approach combines graphical modeling, simulation and model checking to evaluate the security issues of the developed application. Deubler et al. [11] provide a formal analysis of a security-critical service-based software system by leveraging a computer-aided system engineering tool. Best et al. [12] used UMLsec to analyze and explain a search engine used by a German car manufacturer. Further abstract security analysis along with a detailed authentication protocol analysis was conducted on the search engine. These results resulted in static security requirements. Juerjens and Rumm [13] applied UMLsec analysis on the German Health Card Architecture. Using the UMLsec notation enabled the security analyst to characterize their models with respect to the security critical aspects of the system. In [13] Juerjens et al. indicated the effectiveness of the UMLsec deployment approach on mobile communications. They showed that most of the indicated security requirements can be implemented without specifically adapting them to mobile systems. Furthermore they demonstrated that the usage of the UMLsec approach supports the security requirements of mobile communication architectures.
One of the main advocated advantages of MDS is located in the area of model transformation. The system models can be used to directly generate system artifacts such as security policies, access rules, configuration files, and even program code. Alam et al. [14] used models for trust management of Web services and generate XACML (eXtensible Access Control Markup Language) access control policies. Nakamura et al. [15] added security requirements to UML-based application models and transformed them into security configuration files on IBM Web application servers. Jensen and Feja [16] added security requirements to process models on the ARIS platform [2] with the goal to generate security-enhanced business processes for Web services. Souza et al. [17] include security requirements into Business Process Modeling Notation (BPMN) models and transform them into Web Services Business Process Execution Language (WS-BPEL) artifacts for secure service composition in Service-Oriented Architecture (SOA) environments. Menzel and Meinel [18] extended SecureUML to enable secure SOA system design. In a realistic scenario, Ma et al. [19] apply MDS to design and develop a Web service solution for an e-Government system. Bandara et al. [20] conducted a similar evaluation, where they compared the implementation of security patterns with different modeling techniques.

3 Methodology

3.1 Evaluation Scenario

For our evaluation we created a basic web application scenario, which outlines the dangers sketched out by the Open Web Application Security Project (OWASP) in their 2013 published rendition of their TOP 10 [21].

In detail, the situation comprises three machines: A user accesses a web server that is connected with a database server. On the web service, there are two distinct client parts: a regular user and an administrator account. These two have distinctive access permissions in regard to the database server.

[2] `http://www.softwareag.com/corporate/products/aris_platform`

Figure 1 demonstrates this basic scenario. Please note that the model in the figure is modeled with the use of any common modeling language to be formed as unbiased as could reasonably be expected before displaying the situation with the distinctive MDS approaches.

3.2 OWASP Top 10

The OWASP Top 10 [21] provide a comprehensive overview of the most common web application threats and identifies some of the most critical risks. This List of the Top 10 vulnerabilities was created by an online community consisting of domain experts and it is one of the most commonly used reference for web application security analysis. We use this collection as a comparison methodology in order to evaluate the different modeling approaches (Section 5). In this section we give a short overview on the different types of vulnerabilities that are referenced in the OWASP Top 10 of 2013.

A1: Injection An attacker modifies parameters or user input in order to trick the interpreter into executing unintended commands (e.g. SQL, XPath or LDAP queries, operating system commands, we scripting languages, etc.).

A2: Broken Authentication and Session Management Authentication and Session Management have different possible attack vectors including weak passwords, too much information in the displayed error messages, Brute-Force attacks, insecure "Forgotten password" functionality or a broken "Remember me" functionality. These vulnerabilities may lead into Session Hijacking (the takeover of a valid user session).

A3: Cross-Site Scripting (XSS) XSS is possible if user input is sent from the server to the browser without validating or escaping the output data. There are different types of XSS attacks, e.g. Stored XSS (malicious code is persisted in a data storage location), Reflected XSS (malicious code is transmitted via URL to the victim) or DOM (Document Object Model)-based XSS (malicious changes of the DOM tree).

A4: Insecure Direct Object References The attacker changes input parameters to read and modify data of other users (usually, access to this data is denied)

A5: Security Misconfiguration This vulnerability describes common mistakes that harm the security of the system, e.g. sensitive information in the source code, default/install are passwords not changed, errors reveal comprehensive stack traces (information disclosure), security settings of framework not understood and configured properly.

A6: Sensitive Data Exposure This thread includes common problems with sensitive data, e.g. no encryption of sensitive data in transport (e.g. no SSL) or storage (e.g. no password hashing in DB) or usage of weak cryptography (e.g. key generation, key exchange, self-made cryptography or weak hashing).

A7: Missing Function Level Access Control Often applications rely on the user interface layer to perform access control. However this leads to vulnerabilities that are bypassing the user interface (e.g. direct calls of a REST API). An authentication check on the server side is required to prevent any user request without proper authentication.

A8: Cross-Site Request Forgery (CSRF) CSRF describes an attack where a user is undeliberately logged in on a vulnerable website by extracting authentication information from vulnerable sites.

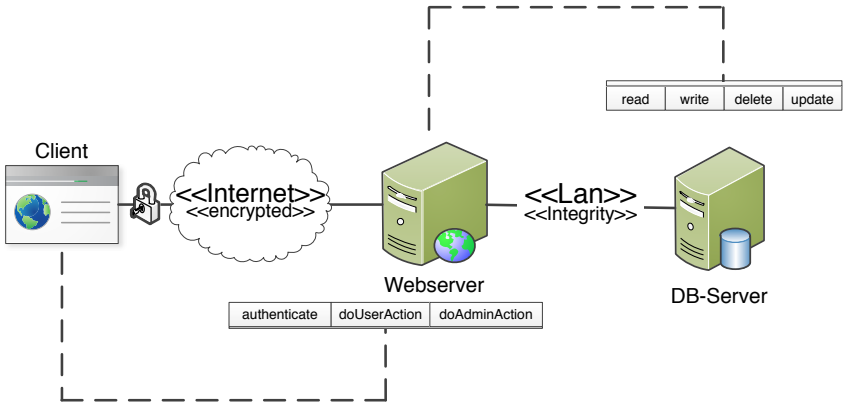


Figure 1: Simple Web Application Scenario

A9: Using Components with Known Vulnerabilities These vulnerabilities are often caused by out-of-date components. To avoid these issues it is necessary to keep book of all used components, version and their dependencies. Furthermore, the security of these components has to be monitored (e.g. public databases or mailing lists) and updated in case of a discovered vulnerability.

A10: Unvalidated Redirects and Forwards A vulnerable website uses redirects, which use user input without sufficient input validation. This enables arbitrary redirects. This attack is similar to XSS attacks (A1) however the impact is less severe. Nevertheless this attack may help an attacker to perform phishing attacks, because the manipulated link targets a trustworthy site but the victim is redirected to the attacker's server.

The scenario of Section 3.1 covers all threats of the OWASP Top 10, because it uses different components and functionalities that can be attacked. The possible threats are mention in the brackets. The web server offers different functionality on a transparent persistence layer (A1, A3, A4, A6, A10). The database server may requires authentication depending on its purpose (A2, A8). The scenario uses a typical web application environment which requires some kind of deployment in an execution environment (A5, A9) and finally, the web server is connected over LAN (Local Area Network) with the database server (A7).

4 Selection of methods

This section evaluates the possibility of applying the different MDS approaches to model the threats outlined in the OWASP Top 10. Each concept is introduced and the web application is then modelled with regard to each of the OWASP threats.

4.1 UML based approaches

UML [22] is a widely used model notation method for constructing and analyzing software system objects. It enables the developer to represent the various parts of a system from the abstract to implementation stage as a series of diagrams. The original UML notations have been extended to allow the inclusion of non-functional system requirements in a direct manner. These non-functional requirements include aspects like security measures and the threat environment. Through the use of extended UML diagrams, a developer can model threats to the system as well as appropriate countermeasures.

4.1.1 SecureUML

SecureUML is an expansion of the standard UML specification that combines the modeling aspect of Role Based Access Control (RBAC) and other security aspects [23]. It is a single purpose extension that enables the modeling of the system with regard to the access control aspects by adding roles, permissions and constraints on a method level to the existing syntax. The designers of SecureUML have created a prototypical tool that allows for the automatic conversion of the model into an EJB (Enterprise Java Beans) based architecture, that combines all standard access controls and primitive comparison functions (e.g. $<, >, \neq \emptyset$). Any remaining capabilities must be implemented by the user. Using these additions, the model can be automatically converted into executable code, helping to mitigate the risks posed by OWASP entries (A1) and (A3). SecureUML derives input validation [24] by implementing a distinct validation class that deals with the input. RBAC is a fundamental part of SecureUML as it ensures the access control restrictions pertaining to objects, databases and files, thus dealing with (A2) and (A4). As RBAC can also be used for URL access restriction, the threat (A7) can be mitigated. The SecureUML specification lacks the functionality to model the transport security aspects and the required logging of queries.

4.1.2 UMLsec

As an expansion to the established UML standard, UMLsec supplies further methods to model the security aspects of software systems based on so-called *secure guards*. This allows for models to be compatible with standard UML diagrams.

When applying the OWASP Top 10 threats to the example web service, there are some aspects that can be prevented with proper UMLsec modeling. The first two threats, Injection (A1) and Cross-Site Scripting (A3), concern the data provided by the user. To prevent attacks on the web service based on this external input we have to check every input. This aspect is modeled with a secure dependency between the web service and the InputValidator, which is called for every input, as one can see in Figure 5. The threats concerning the Broken Authentication and Session Management (A2) cannot be dealt with proper modeling, because the authentication mechanism is encapsulated within the authenticate method. The evaluation of this functionality was omitted, because they are not in the focus of UMLSec. It is possible to model countermeasures against Direct Object Reference (A4), Cross-Site Request Forgery (CSRF) (A8), Missing Function Level Access Control (A7) and Unvalidated Redirects and Forwards (A10) with secure guards. Every possible attack scenario requires dedicated secure guards. One example is a special guard that checks the feasibility of the called method to prevent Cross-Site Request Forgery. Another example is a guard that prevents unauthorized URL or method access. The model (Figure 3), which shows the usage of secure guards, covers our scenario.

UMLsec can be used to tag particular communication paths with security requirements such as encryptions. Besides the aspects that can be modeled with UMLsec, there are some that cannot be modeled by using this technique. Both Security Misconfiguration (A5) and Using Components with Known Vulnerabilities (A9) cannot be modeled by using UMLsec. It is not possible to handle these two issues using model-engineering techniques as these techniques only cover the architecture of the program, not the deployment environment.

4.1.3 Misusecase

Another extension to the use case specification of the UML use case diagram is the misusecase specification. Guttorm Sindre and Andreas L. Opdahl [25] developed this extension to outline potential malicious attacks against a system, which are added to the normal use case diagram with inverted colors. However, it is not possible to provide any tool support to generate code from this use case diagram, due to the

high level of abstraction. The misusecase diagram can be used to combat some of the risks listed in the OWASP Top 10. Attacks such as Injection (A1), Cross-Site Scripting (A3) and Missing Function Level Access Control (A7) can be modeled using the misusecase diagram. Broken Authentication (A2) can be mitigated using the misusecase diagram by modeling unauthorized actions, but any temporal or casual dependencies cannot be modeled. Issues on the OWASP Top 10 relating to configuration: Security Misconfiguration (A6), and those relating to the use of components with known Vulnerabilities (A9) also can not be mitigated using misusecase diagrams.

4.2 Aspect oriented software development

Aspect oriented software development (AOSD) is an emerging approach that looks to promote the advanced separation of concerns. This approach allows system properties such as security to be separately analyzed and then integrated into the system environment.

4.2.1 Aspect oriented modeling

The framework proposed by Zhu et al. [26] is designed to model the potential threats to a system in an aspect-oriented manner. These additions are designed to model an attacker-and-victim relation in the various types of UML diagrams. Due to page limitations, our evaluation only describes the class diagram as it already displays most of the additional features compared to standard UML specifications. The premise of the class diagrams is an abstract *attacker class* that supplies simple attributes and methods. This framework is applicable in the setting of risk-oriented software development. Following a risk analysis of the system, any high impact attacks must be identified and subsequently modeled. The models created in this step can then be converted to aspect-oriented code that can then be integrated into the existing code base. The code generator published by Zhu et al. is equipped to produce both AspectJ and AspectC++ code. These expansions of the standard UML specification are not practical enough to model basic security aspects such as RBAC and transport security. They are only useful for situations involving particular attack scenarios and adding specific countermeasures to a given system. However, it is possible to model each of the aspects listed in the OWASP Top 10 using aspect oriented modeling.

4.2.2 Software Architecture Model (SAM)

Moving away from UML-based modeling techniques, other modeling approaches can be found based on Petri nets and temporal logic, such as the AOD framework described by H.Yu et al. [27]. This framework is designed to model complex workflows and combine them with security aspects. Nodes in the petri net represent single steps of the workflow and the security aspects handle the transitions between each node. The constraints of the workflow are modeled in a temporal logic that enables a formal verification of the system.

4.2.3 Protocol Checker

The AVISPA Tool for automated validation of Internet security protocols and applications mainly covers the verification of (cryptographic) protocols with respect to known attack vectors such as man-in-the-middle and relay attacks. At its core, AVISPA uses a definition language for protocols (HLPSL – High Level Protocol Security Language), which was designed specifically to combine the modeling of protocol flows with security parameters and requirements. Moreover, AVISPA supplies four discrete analysis engines that can be used to work on a problem individually or simultaneously:

- The On-the-fly-Model-Checker (OFMC) uses symbolic model checking for protocol falsification and verification with respect to specified bounds.
- The Constraint-Logic-based-Attack Searcher (CL-AtSe) uses constraint solving for identifying weaknesses and can be extended easily to handle specific algebraic operations.
- The SAT-based Model-Checker (SATMC) generates a propositional formula, which is then fed to a SAT-solver.
- The Tree-Automata-based-on-Automatic-Approximations-for-the-Analysis-of-Security-Protocols module (TA4SP) uses regular tree languages for rewriting and approximating intruder knowledge.

Another tool suited for the analysis of both synchronous and asynchronous protocols is the Symbolic Model Verifier (SMV), which is based on temporal logic. Models are identified in the form of temporal logic formulae, which are then used by the too to specify and handle finite automata and also to verify the validity of the temporal logic formulae. A particular strength of this tool is its ability to handle asynchronous protocols and distributed systems. However, it is not possible to model executable software systems using SMV.

The modeling language Alloy is based on a first-order relational logic, with its main purpose based around modeling software designs. The logical structures of the systems are modeled using relations while the existing properties are modeled using relational operators. Alloy additionally supplies the user with the ability for typing, sub-typing as well as type-checking during runtime as well as the creation of reusable modules. The actual analysis is performed by the tool Alloy Analyzer, which is based on a SAT-solver; due to the construction of the language, the analysis of a model is essentially an application of constraint solving.

It is not practically feasible to use these techniques to model whole software applications as their main goal is to provide a detailed security analysis on the protocol level; they are not concerned with architectural decisions, but rather the execution of protocols using cryptographic primitives. However, they can be useful for the analysis of cryptographic primitives and transport layer protocols and are therefore a good strategy for preventing insufficient transport layer protection.

4.3 Goal driven approaches

Goals cover a range of different types of issues, both functional and non-functional. Goal models exhibit how each of these different high level goals assist others through refinement links down to specific software requirements and environmental assumptions. Functional goals center around the services that are required whereas non-functional goals cover the quality of services such as security or availability.

4.3.1 KAOS

The KAOS model originates from the requirements engineering domain and was designed by researchers at the University of Louvain and the University of Oregon. The name of the methodology KAOS stands for Knowledge Acquisition in autOmated Specification [28] respectively Keep All Objects Satisfied [29] The methodology describes a framework, to model and refine goals as well as the selection of alternatives. The framework is supported by a software solution Objectiver [3], which is developed by a spin-off of the University of Louvain. The software solution supports the developer in designing the goal models and refining these models, as well generating object or operation models. It does not provide any code generation functionality, that transforms the models into actual code. The KAOS model itself starts at a

[3] http://www.objectiver.com

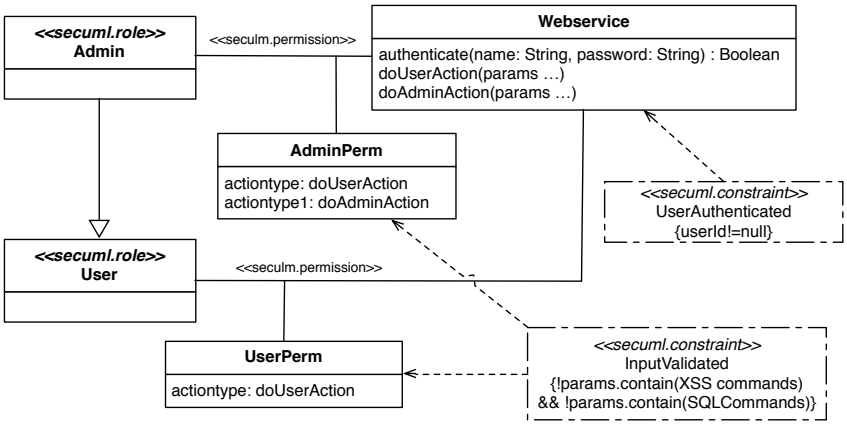


Figure 2: Use case modelled with Secure UML

high level, that describes abstract requirements for the system. These abstract requirements are separated in functional and non-functional requirements, while the security requirements lie in the non-functional section. These goal models can be further used to generate object models, operation models [30] or responsibility models [31] to derive concrete software development requirements and restrictions.

4.3.2 Secure Tropos

The Tropos methodology [32] supports the software development process by describing the environment of the system and the system itself. It is used to model dependencies between different actors who want to achieve different goals by executing plans. There are four different abstraction layers defined to describe different stages of requirements and layers of design. Secure Tropos [33] is an extension to the original Tropos methodology by adding security constraints and secure entities as well as the concepts of ownership, trust and dependency. The Secure Tropos methodology does not allow the designer to model any OWASP TOP 10 threat directly within the model. Nevertheless there are some software solutions, like SecTro2 [4], that support the software engineer during the design and requirements analysis phase.

5 Evaluation

5.1 Secure UML

Beside the intention to use the constraints for both the access restrictions and any pertinent preconditions such as the UserAuthenticated constraint, it is also possible to include more complex requirements to provide input validation as the application of the framework to our use case shows. In Figure 2 we have included the InputValidated constraint, which ensures that the parameters do not feature any strings that could be exploited either by Cross-Site Scripting or SQL injection. This additional functionality to cover Cross-Site Scripting and SQL injection checks must be implemented by the user as the tool only allows for primitive comparison functionalities. Furthermore, the Secure UML specification does not provide the functionality to model both transport security and the logging of database queries.

[4] `http://sectro.securetropos.org`

5.2 UMLsec

This evaluation focuses on the class and the deployment diagram, because these two diagrams already cover all security requirements of our simple web application scenario.

The feature of transport security tangles the communication between the three components, as shown by Figure 1. The communication between the client and the application server is sent via the Internet, thus all service-calls and the resulting replies must be encrypted. The communication between the application server and the database server is less important because they are both connected within the same local network. This allows for the reduction of the security requirement from the encryption level to the integrity level. Both of these stereotypes are expressed using the UMLsec specification. Both the Internet and LAN environments are added to the link between the systems and the calls are tagged with the required stereotype. Given the heterogeneous nature of the systems and the ease with which the transport requirements can be modeled, it is not feasible to automatically generate code to ensure compliance with the requirements.

The aspects relating to both *authentication* and *RBAC* are modeled within the class diagram. The basic model has to be expanded to include two further classes (UserAction and AdminAction) to define user specific access control as the UMLsec specification only supports class based access restrictions. Both of these classes are simple wrapper classes, annotated with two different guards. These guards are both called from the web service class and verify whether or not the current user has a specific role, assigned to them by a successful authentication.

The model (Figure 3) that demonstrates the usage of secure guards shows this scenario. Due to this implicit mechanism, additional constraints, such as the users have to be authenticated, do not have to be modeled. A successful evaluation of how UMLsec properties can be converted into code is shown in [34]. One disadvantage to this style of modeling is that it often fails to scale sufficiently for additional roles and it also increases the complexity of the model. The correct input validation is modeled using a secure dependency between the web service and the InputValidator, which is called for every input, as demonstrated in Figure 4. These guards verify whether the users have sufficient privileges to perform a given action.

The last aspect to be looked at is the claim that all queries are logged. This aspect in modeled with the secure dependency addition of UMLsec. By means of this addition, it is possible to model the constraint with the log method of the logger class. As each user could potentially submit malicious input to the system in our scenario, there has to be some instance of input validation to mitigate the risk of both SQL injection and Cross Site Scripting attacks. This aspect was modeled using the secure dependency addition: Each input that is passed on to a method provided by the web service must be checked for any malicious input (see Figure 5).

5.3 Misusecase

The use case diagram shown in Figure 6 demonstrates the modeling of different threats to the system. Any threats carried out by an attacker are shown with a normal use case actor marked with a black background color.

This also applies to the misusecases shown in the diagram, which are represented by ordinary use case elements marked with a black background. The misusecase diagram allows for the modeling of high-level threats that can be executed by different actors within the system, but does not provide the functionality to model any potential countermeasures or mitigation approaches. The only available method to model countermeasures is to extend the existing use cases to implement organizational countermeasures, such as additional permission checks.

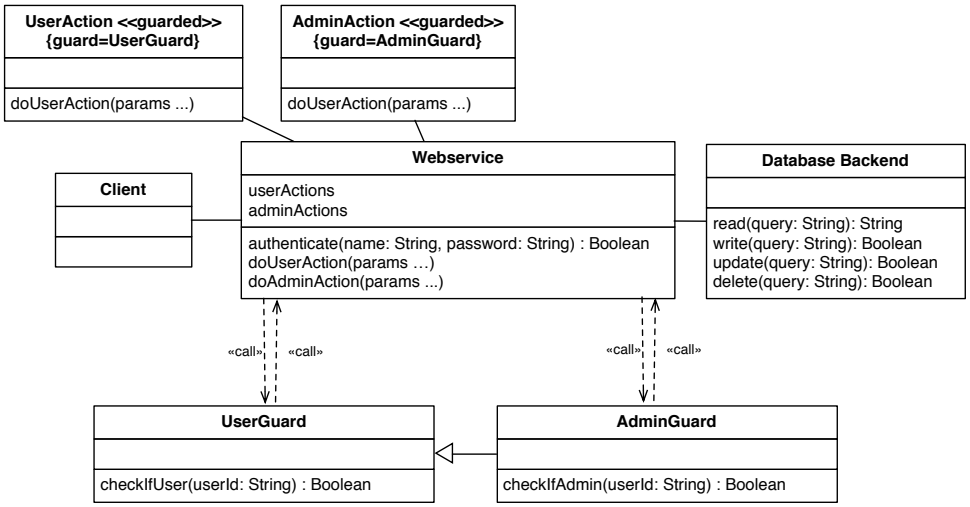


Figure 3: Secure Guards in UMLsec

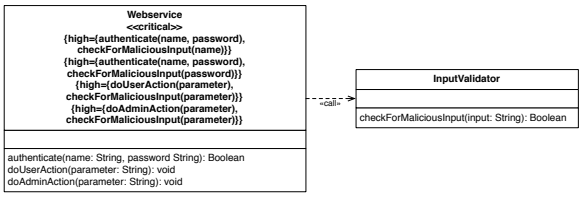


Figure 4: Input Validation in UMLsec

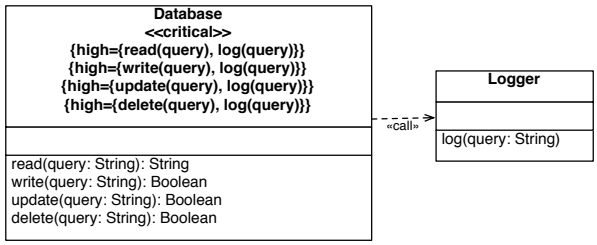


Figure 5: Secure Dependency in UMLsec

5.4 Aspect oriented modeling

The basis of the class diagrams is an abstract Attacker class that provides the basic attributes and methods for the concrete attacker class. In our example, the concrete attacker tries to tamper the authentication using invalid input (Figure 7. This attack is modeled as an aspect that provides some methods to execute checks to prevent this attack. The remaining part of the diagram is a simplified representation of our

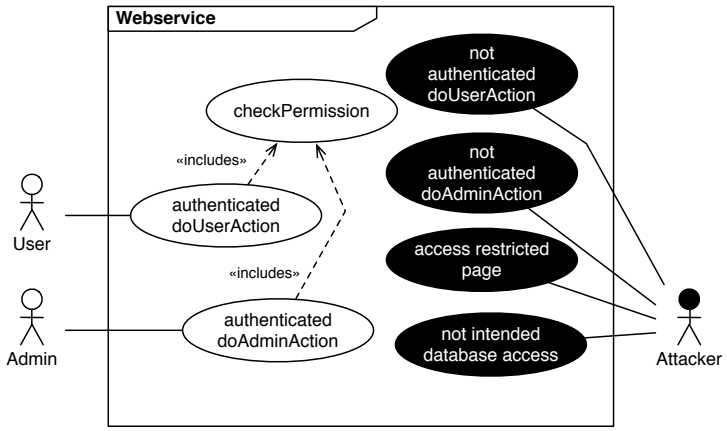


Figure 6: Modeling of malicious acts with misusecase diagrams

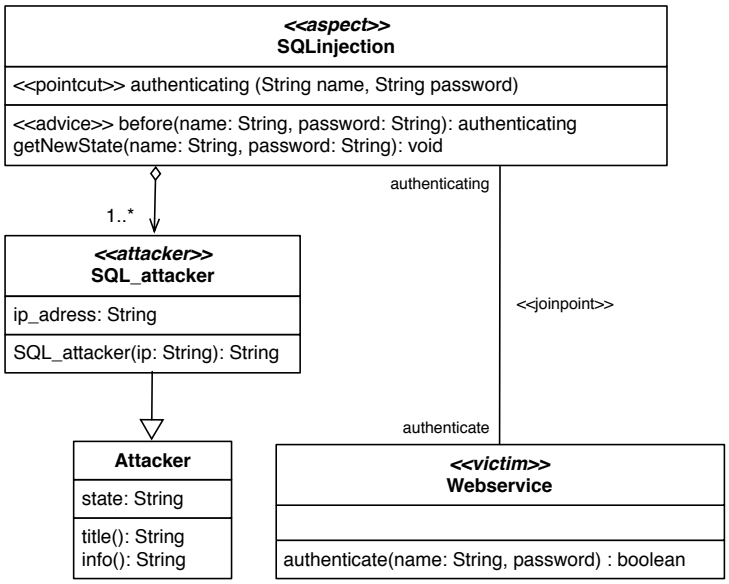


Figure 7: Aspect Oriented Modeling

basic UML class diagram. In the context of this framework it is feasible to omit all classes or methods that are not used in this attack. Every diagram that is modeled within this framework visualizes only one attack. In a real world setting there will be numerous different diagrams that model different attacks on a given system.

In general it is possible to model all aspects of the OWASP Top10 using aspect oriented modeling. Still, the approach is only feasible for covering the most pressing topics like injections and cross site scripting. In aspect oriented modeling every possible attack needs to be modeled with respect to its effects on the system, which implies that all possible attacks need to be known beforehand. Furthermore, in case of real-life-size applications, the number of possible attack scenarios that need to be modeled separately will grow drastically.

5.5 SAM

Due to the lack of complex workflows in our scenario, we omitted a detailed analysis of this framework. The single method calls do not trigger any workflows within the web service. Currently there is no tool support for this framework that provides automatic code generation, but this framework can be used in order to perform a detailed risk analysis of a complex workflow.

5.6 KAOS

The KAOS model itself begins at a high level and describes the abstract requirements of the system. The security requirements lie in the non-functional section, as shown in Figure 8. Figure 9 shows a refinement of the secure system requirement, where the majority of the OWASP Top 10 issues can be modeled. Figure 10 demonstrates a model for a concrete requirements model for calling the doAdminAction method. This model already includes actors and specific requirements related to the rather high-level requirements such as restricted access or authenticity.

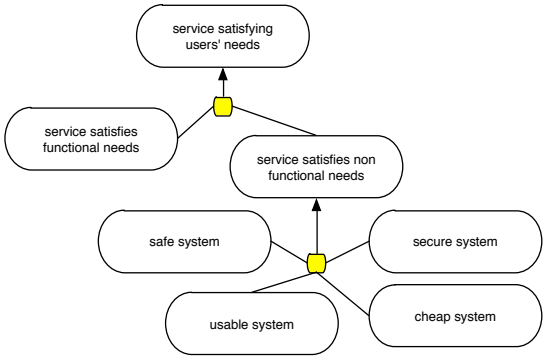


Figure 8: Basic goal model

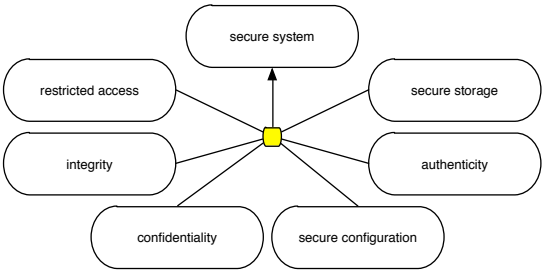


Figure 9: Refined goal model

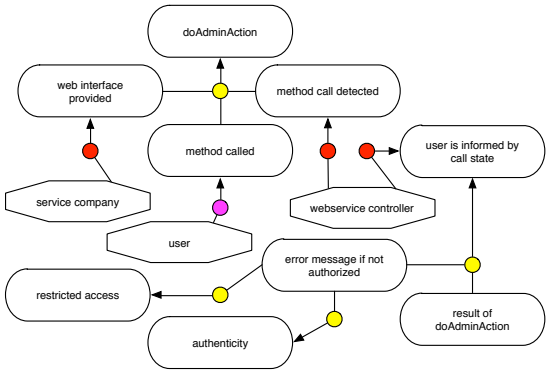


Figure 10: Goal model for a specific action

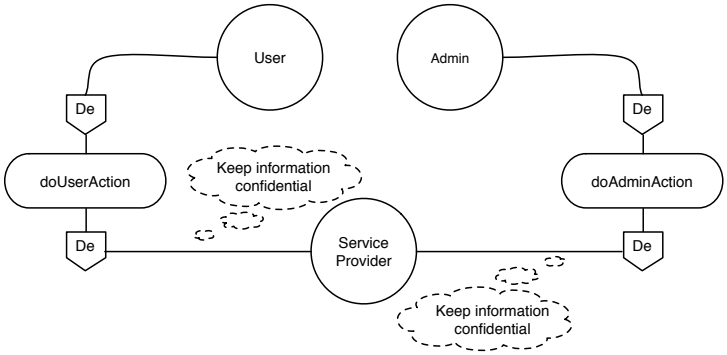


Figure 11: Security constraints modeled with Tropos

5.7 Secure Tropos

Figure 11 demonstrates a basic dependency model with security constraints, which are modeled in the cloud shaped elements. It shows each of the three actors within the system, the user, admin and service provider as well as the two plans available to be executed by the user and admin actors. The security constraints can be used to introduce requirements for actions between the user and admin actors but not for the systems, such as the database server. This methodology is used to model the dependencies and trust relations between multiple stakeholders. However, it is not practical to apply this methodology to our evaluation scenario to improve the security of the system.

6 Discussion

6.1 Comparing results

In the following section we will give an overview on the capabilities of the different modelling approaches and compare them with respect to the OWASP Top Ten of 2013.

OWASP Top 10	Secure UML	UMLsec	Misuse-case	Aspect Oriented	KAOS	Protocol Checker	Secure Tropos
Injection (A1)	✓	✓	✓	✓	✓	✗	✗
Broken Auth. and Session Mgmnt. (A2)	✓	✗	✓	✓	✓	✗	✗
XSS (A3)	✓	✓	✓	✓	✓	✗	✗
Insecure Direct Object Ref. (A4)	✓	✓	✓	✓	✓	✗	✗
Security Misconfiguration (A5)	✗	✗	✗	✓	✓	✗	✗
Sensitive Data Exposure (A6)	✗	✗	✗	✓	✓	✗	✗
Missing Function Level Access Control (A7)	✓	✓	✓	✓	✓	✗	✗
CSRF (A8)	✗	✓	✓	✓	✓	✗	✗
Using Components with Known Vulnerabilities (A9)	✗	✗	✗	✓	✓	✗	✗
Unvalidated Redirects and Forwards (A10)	✗	✓	✓	✓	✓	✗	✓
Toolsupport	✓	✗	✗	✓	✓	✗	✓

Table 1: Summary of OWASP Top 10 mitigation coverage

Table 1 provides a summary on the capabilities of the modeling techniques outlined and compared in this paper. The approaches can be divided into four distinct categories that they focus on:

Architecture: The UML-based approaches work on an architectural level, thus features like RBAC can easily modelled (see [35]). With RBAC, the OWASP threats ”Broken authentication and session management” (A2, limited in UMLsec)), ”Insecure direct object reference” (A4) and ”Missing function level access control” (A7) can be mitigated, thus this feature is passed on to the UML-based approaches. Due to the existence of tools for automated code generations, the threats ”Injection” (A1) and ”XSS” (A3) can be mitigated too. Since deployment is not part of the architecture, threats relating to this part of the development process, specifically threats A5 and A9, cannot be covered with these tools. The main differences between the architectural approaches are based on differences in their design and are discussed in Section 4.1.

Behaviour: Misusecase on the other side focusses on modelling behaviour. This modelling works only with respect to internal assets, allowing for flexible modelling of issues like access control, still, (behavioural) external assets like deployment, exposure of sensitive data or using secure components cannot be modelled.

Communication/Protocols: Protocol checkers focus on the evaluation of (security) protocols, omitting the architectural layer. Network and transport layer security can thus be modelled easily, still, those approaches fail at actually defining the assets in an architecture. This makes it impossible to model injections, exposure of sensitive data CSRF, XSS or access control, or even more high-level assets relating to the actual implementation or deployment issues (e.g. "Using components with known vulnerabilities" (A9)).

Risk and Requirement modelling: Aspect oriented approaches as well as KAOS are focussed towards extensive and very detailed risk analysis and modelling of specific requirements. While these solutions are very powerful in theory, they lack practicability, e.g. in the aspect oriented approach, all possible attacks have to be recognized, analyzed and modelled.

6.2 Current Application and Combination of Methods

In order to mitigate the deficiencies of the individual modelling approaches, the typical secure software lifecycle is comprised of several steps, involving several different models and abstractions. Starting with a detailed requirements analysis a preliminary architecture is modelled. In addition external requirements, especially concerning the definition of sensitive data sets, are introduced as side parameters. Based on these conceptual designs, a risk analysis is conducted, the results are used as input for further changes in the underlying architecture. While this modelling covers the architectural level, other issues are delegated: Secure coding guidelines enforce the non-usage of components with known vulnerabilities (A9), as well as secure deployment guidelines try to mitigate security misconfigurations (A5). Furthermore, the very important topic of using and designing secure protocols is delegated: While the need for secure protocols can be annotated in the architectural view, the actual security of the protocols is ensured by external protocol checkers, or even more advanced techniques like language security or cryptoanalysis.

7 Conclusions

The modeling of countermeasures and mitigation of the OWASP TOP 10 threats are supported in most of the UML based modeling methodologies, mostly by adding additional constrains on an implementation level. In contrast the misuse case diagram and the goal-based approach do not model the implementation. Instead they use a model of a higher abstraction layer that shows real world interactions and requirements. With these high level requirements some threats can be described as used in the KAOS methodology. Therefore mitigation of the modeled threats cannot be used to identify potential security issues or potential collisions for conflicting goals. The detection of conflicts and their resolution are crucial in large systems with different stakeholders involved who have conflicting requirements. In addition we showed that model driven engineering does not improve the security in general by adding implicit mitigation procedures or checks for potential flaws in the models, like those listed in the OWASP Top 10. These methodologies are intended to only support the developers by indicating possible locations of conflicts. This can be achieved by goal based methodologies or by the addition of standard mitiga-

tion features to existing systems like UMLsec and Secure UML methodologies. Table 1 presents an summarized overview of the capabilities of the evaluated methodologies.

In conclusion model driven engineering can reduce potential threats that are listed in the OWASP Top 10 indicting them in the model. However this indication does not ensure that the software architect, who is in charge of designing the model, plans appropriate countermeasures or mitigation features and further the actual implementation is compliant with the model itself.

Acknowledgements

This work has been supported by the Austrian Research Promotion Agency (FFG) under the Austrian COMET Program.

References

[1] J. Jürjens, "UMLsec: Extending UML for secure systems development," in *Proc. of the 5th International Conference on the Unified Modeling Language (UML'02), Dresden, German, LNCS*, vol. 2460. Springer-Verlag, September-October 2002, pp. 412–425.

[2] T. Lodderstedt, D. Basin, and J. Doser, "SecureUML: A UML-based modeling language for model-driven security," in *Proc. of the 5th International Conference on the Unified Modeling Language (UML'02), Dresden, German, LNCS*, vol. 2460. Springer-Verlag, September-October 2002, pp. 426–441.

[3] C. Hochreiner, Z. Ma, P. Kieseberg, S. Schrittwieser, and E. Weippl, "Using model driven security approaches in web application development," in *Proc. of the 2nd IFIP TC5/8 Information & Communication Technology-EurAsia Conference (ICT-EurAsia'14), Bali, Indonesia, LNCS*, vol. 8407. Springer-Verlag, April 2014, pp. 419–431.

[4] D. Basin, M. Clavel, and M. Egea, "A decade of model-driven security," in *Proc. of the 16th ACM symposium on Access control models and technologies (SACMAT'11), Innsbruck, Austria*. ACM, June 2011, pp. 1–10.

[5] J. Jürjens, *Secure systems development with UML*. Springer-Verlag, October 2004.

[6] D. Basin, J. Doser, and T. Lodderstedt, "Model driven security for process-oriented systems," in *Proc. of the 8th ACM symposium on Access control models and technologies (SACMAT'03), Villa Gallia, Como, Italy*. ACM, June 2003, pp. 100–109.

[7] D. Basin, J. Doser, and T. Lodderstedt, "Model driven security: From uml models to access control infrastructures," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 15, no. 1, pp. 39–91, January 2006.

[8] K. Kasal, J. Heurix, and T. Neubauer, "Model-driven development meets security: An evaluation of current approaches," in *Proc. of the 44th Hawaii International Conference on System Sciences (HICSS'11), Manoa, Hawaii, USA*. IEEE, January 2011, pp. 1–9.

[9] J. Lloyd and J. Jürjens, "Security analysis of a biometric authentication system using umlsec and jml," *Model Driven Engineering Languages and Systems*, pp. 77–91, October 2009.

[10] J. Grünbauer, H. Hollmann, J. Jürjens, and G. Wimmel, "Modelling and verification of layered security protocols: A bank application," in *Proc. of 22nd International Conference on Computer Safety, Reliability, and Security (SAFECOMP'03), Edinburgh, UK, LNCS*, vol. 2788. Springer-Verlag., September 2003, pp. 116–129.

[11] M. Deubler, J. Grünbauer, J. Jürjens, and G. Wimmel, "Sound development of secure service-based systems," in *Proc. of the 2nd international conference on Service oriented computing (ICSOC'04), New York City, New York, USA*. ACM, November 2004, pp. 115–124.

[12] B. Best, J. Jurjens, and B. Nuseibeh, "Model-based security engineering of distributed information systems using umlsec," in *Proc. of the 29th International Conference on Software Engineering (ICSE'07), Minneapolis, Minnesota, USA*. IEEE, May 2007, pp. 581–590.

[13] J. Jurjens, J. Schreck, and P. Bartmann, "Model-based security analysis for mobile communications," in *ACM/IEEE 30th International Conference on Software Engineering (ICSE'08), Cape Town, South Africa*. IEEE, May 2008, pp. 683–692.

[14] M. Alam, R. Breu, and M. Hafner, "Model-driven security engineering for trust management in sectet," *Journal of Software*, vol. 2, no. 1, pp. 47–59, January 2007.

[15] Y. Nakamura, M. Tatsubori, T. Imamura, and K. Ono, "Model-driven security based on a web services security architecture," in *Proc. of the 2005 IEEE International Conference on Services Computing (SCC'05), Orlando, Florida, USA*, vol. 1. IEEE, July 2005, pp. 7–15.

[16] M. Jensen and S. Feja, "A security modeling approach for web-service-based business processes," in *Proc. of the 16th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems (ECBS'09), San Francisco, California, USA*. IEEE, April 2009, pp. 340–347.

[17] A. Souza, B. Silva, F. Lins, J. Damasceno, N. Rosa, P. Maciel, R. Medeiros, B. Stephenson, H. Motahari-Nezhad, J. Li *et al.*, "Incorporating security requirements into service composition: From modelling to execution," in *Proc. of the 7th International Joint Conference on Service-Oriented Computing (ICSOC-ServiceWave'09), Stockholm, Sweden, LNCS*, vol. 5900. Springer-Verlag, November 2009, pp. 373–388.

[18] M. Menzel and C. Meinel, "Securesoa modelling security requirements for service-oriented architectures," in *Proc. of the 7th IEEE International Conference on Services Computing (SCC'10), Miami, Florida, USA*. IEEE, July 2010, pp. 146–153.

[19] Z. Ma, C. Wagner, and T. Bleier, "Model-driven security for web services in e-government system: Ideal and real," in *Proc. of the 7th International Conference on Next Generation Web Services Practices (NWeSP'11), Salamanca, Spain*. IEEE, October 2011, pp. 221–226.

[20] A. Bandara, H. Shinpei, J. Jurjens, H. Kaiya, A. Kubo, R. Laney, H. Mouratidis, A. Nhlabatsi, B. Nuseibeh, Y. Tahara *et al.*, "Security patterns: Comparing modeling approaches," October 2010.

[21] OWASP, "Open web application security project top 10," https://www.owasp.org/index.php/Top10#OWASP_Top_10_for_2013 (Last Access: Apr 23, 2014).

[22] J. Rumbaugh, I. Jacobson, and G. Booch, *The Unified Modeling Language Reference Manual (2nd Edition)*. Pearson Higher Education, May 2004.

[23] R. Sandhu, E. Coyne, H. Feinstein, and C. Youman, "Role-based access control models," *Computer*, vol. 29, no. 2, pp. 38–47, February 1996.

[24] P. Hayati, N. Jafari, S. Rezaei, S. Sarenche, and V. Potdar, "Modeling input validation in uml," in *Proc. of the 19th Australian Conference on Software Engineering (ASWEC'08), Perth, Australia*. IEEE, March 2008, pp. 663–672.

[25] G. Sindre and A. Opdahl, "Templates for misuse case description," in *Proc. of the 7th International Workshop on Requirements Engineering, Foundation for Software Quality (REFSQ'01), Interlaken, Switzerland*, June 2001.

[26] Z. Zhu and M. Zulkernine, "A model-based aspect-oriented framework for building intrusion-aware software systems," *Information and Software Technology*, vol. 51, no. 5, pp. 865–875, May 2009.

[27] H. Yu, D. Liu, X. He, L. Yang, and S. Gao, "Secure software architectures design by aspect orientation," in *Proc. of the 10th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS'05), Shanghai, China*. IEEE, June 2005, pp. 47–55.

[28] A. van Lamsweerde, A. Dardenne, B. Delcourt, and F. Dubisy, "The KAOS project: Knowledge acquisition in automated specification of software," in *Proc. of the 1991 AAAI Spring Symposium Series, Track: Design of Composite Systems, Stanford University, USA*, November 1991, pp. 59–62.

[29] A. Van Lamsweerde and E. Letier, "From object orientation to goal orientation: A paradigm shift for requirements engineering," in *Proc. of the 9th International Workshop on Radical Innovations of Software and Systems Engineering in the Future (RISSEF'02), Venice, Italy, LNCS*, vol. 2941. Springer-Verlag, October 2004, pp. 153–166.

[30] E. Letier and A. Van Lamsweerde, "Deriving operational software specifications from system goals," *ACM SIGSOFT Software Engineering Notes*, vol. 27, no. 6, pp. 119–128, November 2002.

[31] E. Letier and A. Van Lamsweerde, "Agent-based tactics for goal-oriented requirements elaboration," in *Proc. of the 24th International Conference on Software Engineering (ICSE'02), Orlando, Florida, USA*. ACM,

May 2002, pp. 83–93.

[32] P. Bresciani, A. Perini, P. Giorgini, F. Giunchiglia, and J. Mylopoulos, "Tropos: An agent-oriented software development methodology," *Autonomous Agents and Multi-Agent Systems*, vol. 8, no. 3, pp. 203–236, May 2004.

[33] H. Mouratidis and P. Giorgini, "Enhancing secure tropos to effectively deal with security requirements in the development of multiagent systems," in *Safety and Security in Multiagent Systems - Research Results from 2004-2006, LNCS*, M. Barley, H. Mouratidis, A. Unruh, D. Spears, P. Scerri, and F. Massacci, Eds. Springer-Verlag, October 2009, vol. 4324, pp. 8–26.

[34] L. Montrieux, J. Jürjens, C. Haley, Y. Yu, P. Schobbens, and H. Toussaint, "Tool support for code generation from a umlsec property," in *Proc. of the 25th IEEE/ACM International Conference on Automated Software Engineering (ASE'10), Antwerp, Belgium*. ACM, September 2010, pp. 357–358.

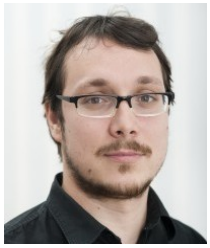
[35] M. Strembeck and J. Mendling, "Modeling process-related RBAC models with extended uml activity models," *Information and Software Technology*, vol. 53, no. 5, pp. 456–483, May 2011.

Author Biography

Christoph Hochreiner is a researcher at SBA Research the Austrian non-profit research institute for IT-Security. Christoph's research interests cover the whole spectrum of privacy preserving mechanism, especially in the area of location privacy and model driven security.

Peter Frühwirt is a researcher at SBA Research and lecturer at the Vienna University of Technology. Peter received a Dipl. Ing. (equivalent to MSc) degree in Software Engineering and Internet Computing in 2013. His research interests include mobile security and database forensics.

Zhendong Ma works as a research scientist at the Austrian Institute of Technology (AIT). He conducts application-oriented research in the area of computer and information security, in which he applies theoretical research work to solve real world problems as well as identifies and conceptualizes research problems from the practice. He holds a doctorate degree from University of Ulm, Germany, where he worked in the area of privacy and security of vehicular communications. Currently he is involved in national and EU projects on critical infrastructure protection, cloud security, digital identity, and privacy of surveillance infrastructure.



Peter Kieseberg received a master's degree in Technical Mathematics in Computer Science from the Vienna University of Technology with specializations in cryptography and numerical mathematics. He worked as a consultant in the telecommunication sector for several years before joining SBA Research.

Sebastian Schrittwieser is a lecturer and researcher at the University of Applied Sciences St. Pölten, Austria. He received his doctorate degree with distinction from Vienna University of Technology in 2014. Sebastian's research interests include, among others, digital forensics, software protection, code obfuscation, and mobile security.

Edgar R. Weippl is Research Director of SBA Research and associate professor (Privatdozent) at the Vienna University of Technology (CISSP, CISA, CISM, CRISC, CSSLP, CMC). His research focuses on applied concepts of IT-security; he organizes the ARES conference and is on the editorial board of Elsevier's Computers & Security journal (COSE).