Model-based Security Engineering of Distributed Information Systems using UMLsec

Bastian Best
BMW Group
München, Germany
Bastian.Best@bmw.de

Jan Jürjens* and Bashar Nuseibeh
Department of Computing, The Open University
Walton Hall, Milton Keynes, MK7 6AA, UK
http://www.jurjens.de/jan – B.Nuseibeh@open.ac.uk

Abstract

Given the explosive growth of digitally stored information in modern enterprises, distributed information systems together with search engines are increasingly used in companies. By enabling the user to search all relevant information sources with one single query, however, crucial risks concerning information security arise. In order to make these applications secure, it is not sufficient to penetrate-and-patch past system development, but security analysis has to be an integral part of the system design process for such distributed information systems. This work presents the experiences and results of the security analysis of a search engine in the intranet of a German car manufacturer, by making use of an approach to Model-based Security Engineering that is based on the UML extension UMLsec. The focus lies on the application's single-sign-on-mechanism, which was analyzed using the UMLsec method and tools. Main results of the paper include a field report on the employment of the UMLsec method in an industrial context as well as indications on its benefits and limitations.

1. Introduction

The extensive use of electronic communication technologies such as e-mail and the networking of web-based information sources in corporate intranets has led to an explosive growth of digitally stored information. An approach to prevent information overflow in this huge information supply is the provision of personalized information supplies. However, this approach carries critical risks concerning information security. In order to address these risks and enable secure information management, security analysis has to be embedded into the early phases of system design.

This work presents the results of the security analysis of a corporate meta search engine in the intranet of a German car manufacturer. The security critical parts of the system were analyzed using UMLsec [5], a UML extension which allows the application developer to embed security related information into system design, as well as to conduct security analyses on the model layer. The goal of this work was to gain experiences in the use of the UMLsec method in an industrial context and to show its benefits and limitations.

Empirical studies on the use of model-based development techniques in software development in general have so far unfortunately been limited. [11] reports on an attempt in that direction and the problems encountered while [10] discusses the challenges faced by empirical software engineering in general. A very interesting report on using UMLsec in industrial development of security-critical software can be found in [1]. Another case study employing a method close to the UMLsec approach in an industrial context is [4], reporting on a project with a major German bank. [14] reports on a student project using a security extension of the AutoFocus tool (that was developed jointly and in parallel with the UMLsec extension) to develop a mobile payment application. Although this paper does not aim to be a complete or controlled empirical study, we hope to contribute to filling the existing gap on empirical validation of model-based development of software, with a specific focus on the security-critical software domain.

The paper is structured as follows. In Section 2 we briefly introduce the required elements of the UMLsec notation. In Section 3 we give an overview of the application under consideration, a meta search engine with a single-sign-on feature. The authentication protocol for the login process is modeled and a detailed security analysis is carried out in Section 4. We discuss some lessons learned from the case study in Section 5, and finally end with a conclusion indicating further work.

[*] This work has been performed while this author was still at Software & Systems Engineering, Technische Universität München.

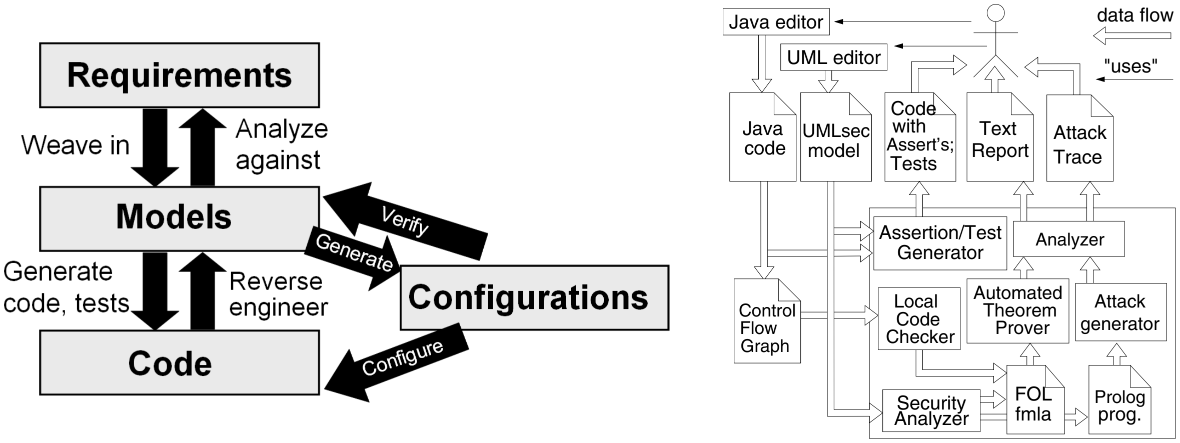


Figure 1. a) Model-based Security Engineering; b) Model-based Security Tool Suite

2. Model-based Security Engineering using UMLsec

Model-based Security Engineering (MBSE, [5, 6, 9, 7]) provides a soundly based approach for developing security-critical software where recurring security requirements (such as secrecy, integrity, authenticity and others) and security assumptions on the system environment, can be specified either within a UML specification, or within the source code (Java or C) as annotations (cf. Fig. 1a). Various analysis plugins in the associated UMLsec tool framework [13] (Fig. 1b) generate logical formulas formalizing the execution semantics and the annotated security requirements. Automated theorem provers and model checkers automatically establish whether the security requirements hold. If not, a Prolog-based tool automatically generates an attack sequence violating the security requirement, which can be examined to determine and remove the weakness. This way we encapsulate knowledge on prudent security engineering as annotations in models or code and make it available to developers who may not be security experts. Since the analysis that is performed is too sophisticated to be done manually, it is also valuable to security experts.

One can use MBSE within model-based development (Fig. 1a). Here one first constructs a model of the system. Then, the implementation is derived from the model: either automatically using code generation, or manually, in which case one can generate test sequences from the model to establish conformance of the code regarding the model. The goal is to increase the quality of the software while keeping the implementation cost and the time-to-market bounded. For security-critical systems, this approach allows one to consider security requirements from early on in the development process, within the development context, and in a seamless way through the development cycle: One can first check that the system fulfills the relevant security requirements on the design level by analyzing the model and secondly that the code is in fact secure by generating test sequences from the model. However, one can also use our analysis techniques and tools within a traditional software engineering context, or where one has to incorporate legacy systems that were not developed in a model-based way. Here, one starts out with the source code. Our tools extract models from the source code, which can then again be analyzed against the security requirements. Using MBSE, one can incorporate the configuration data (such as user permissions) in the analysis, which is very important for security but often neglected.

Part of the MBSE approach is the UML extension UMLsec for secure systems development which allows the evaluation of UML specifications for vulnerabilities using a formal semantics of a simplified fragment of the UML. The UMLsec extension is given in form of a UML profile using the standard UML extension mechanisms. *Stereotypes* are used together with *tags* to formulate the security requirements and assumptions. *Constraints* give criteria that determine whether the requirements are met by the system design, by referring to a precise semantics of the used fragment of UML. The security-relevant information added using stereotypes includes security-relevant information covering the following aspects:

- Security assumptions on the physical system level, for example the stereotype «`encrypted`», when applied to a link in a UML deployment diagram, states that this connection has to be encrypted.
- Security requirements on the logical level, for example related to the secure handling and communication of data, such as «`secrecy`» or «`integrity`».
- Security policies that system parts are required to obey, such as «`fair exchange`» or «`data security`».

The UMLsec tool-support in Fig. 1b can then be used to check the constraints associated with UMLsec stereotypes mechanically, based on XMI output of the diagrams from the UML drawing tool in use [13, 6]. There is also a framework for implementing verification routines for the constraints associated with the UMLsec stereotypes. Thus advanced users of the UMLsec approach can use this framework to implement verification routines for the constraints of self-defined stereotypes. The semantics for the fragment of UML used for UMLsec is defined in [5] using so-called *UML Machines*, which is a kind of state machine with input/output interfaces and UML-type communication mechanisms. On this basis, important security requirements such as secrecy, integrity, authenticity, and secure information flow are defined. To support stepwise development, it has been showed that secrecy, integrity, authenticity, and secure information flow are *preserved* under refinement and the composition of system components. The approach also supports the secure development of layered security services (such as layered security protocols). UMLsec can be used to specify and implement security patterns, and is supported by dedicated secure systems development processes, in particular an Aspect-Oriented Modeling approach which separates complex security mechanisms (which implement the security aspect model) from the core functionality of the system (the primary model) in order to allow a security verification of the particularly security-critical parts, and also of the composed model. For more information on the usage and semantics of UMLsec the reader is referred to [5].

3. A Model of the Meta Search Engine

The application under consideration is a meta search engine in the corporate intranet of a german car manufacturer, subsequently called *Metasearch*. The main purpose of Metasearch is to give a single point of access for employees to search in all relevant information sources (internal and external) with a single query. Therefore, it provides personalization capabilities, i.e. a user can store his preferred information sources including passwords for restricted sources. The included sources range from non-confidential information such as enterprise announcements, but also confidential documents such as meeting protocols or even development specifications of car components and design specifications. Since every employee should be able to use MetaSearch (depending on his access level of course), there are more than 1,000 potential users of the application. The goal is to index about 280,000 documents and allow about 20,000 queries per day.

One requirement for Metasearch therefore was its seemless integration into an existing enterprise-wide security reference architecture. This architecture defines the security requirements applications have to fulfill. Furthermore, it provides various security-related services, which new applications can use. This includes the fields of *user authentication*, management of *user roles and rights* as well as building blocks for the hook-up of new security-critical applications within this framework. Metasearch had to be integrated into a global single-sign-on infrastructure, so that a user does not have to provide his credentials if he logged in to a federated application beforehand.

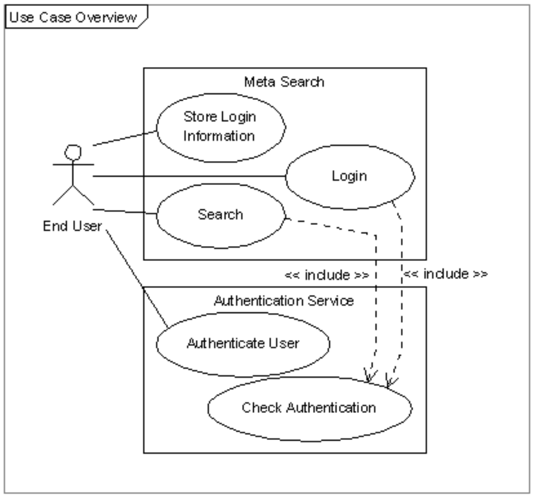


Figure 2. Security-critical use cases of Metasearch

Overview of the Application

Focussing on the security critical parts of Metasearch, three main use cases were identified, shown in figure 2. Integration of Metasearch into a global single sign on infrastructure is implemented in the use case `Login`, which will be described in more detail below. The use case `Store Login Information` denotes the functionality of storing the users' credentials for access to restricted information sources. The use case `Search` handles the main functionality of Metasearch, namely searching the selected sources with a specific query. The specification of the latter two use cases are omitted for the sake of brevity and are described in more detail in [2].

The UML Subsystem *Login*

In order to analyze the security aspects of a system model, the different models are combined in a UML subsystem. The UML subsystem `Login` is comprised of a dynamic model describing the functionality of the use case (in the form of a UML sequence diagram) as well as static models specifying the interacting entities (in the form of a UML class diagram) and their physical allocation (in form of a

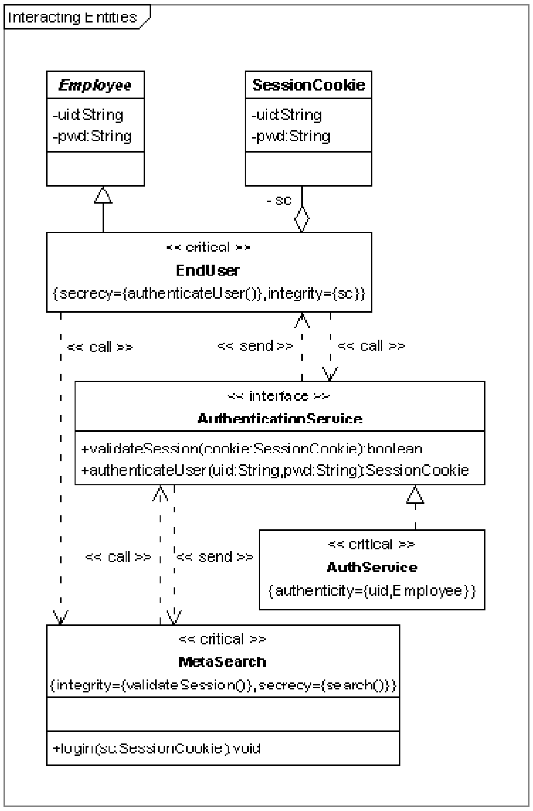

Interacting Entities
Employee
-uid:String
-pwd:String
SessionCookie
-uid:String
-pwd:String
- sc
<< critical >>
EndUser
{secrecy={authenticateUser()},integrity={sc}}
<< call >>
<< send >>
<< call >>
<< interface >>
AuthenticationService
+validateSession(cookie:SessionCookie):boolean
+authenticateUser(uid:String,pwd:String):SessionCookie
<< call >>
<< send >>
<< critical >>
AuthService
{authenticity={uid,Employee}}
<< critical >>
MetaSearch
{integrity={validateSession()},secrecy={search()}}
+login(sc:SessionCookie):void

Figure 3. Class structure of the use case Login

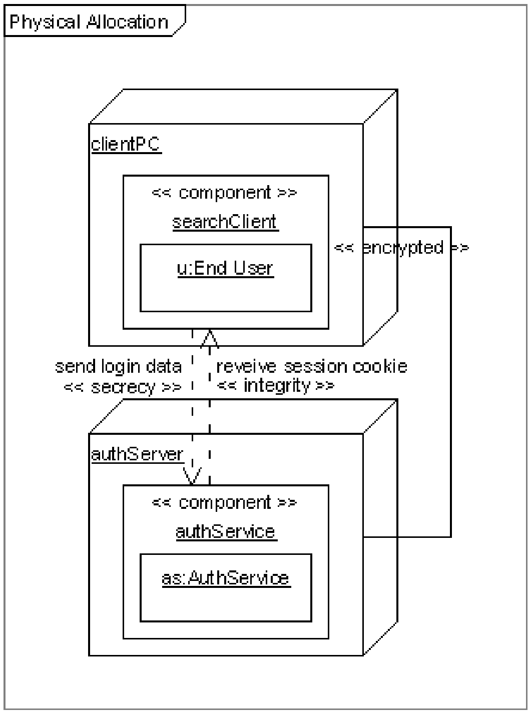


Figure 5. Physical Allocation

UML deployment diagram). Furthermore, a formal attacker model is provided.

Interacting Entities The objects interacting in the use case `Login` are shown in figure 3. The central classes are `EndUser`, which represents the user who wants to log in, and `AuthService`, which represents a central authentication server offering services to validate a user session and to authenticate a user through the interface *`AuthenticationService`*. In order to login, `EndUser` invokes the method `authenticateUser()` providing his credentials (his user-ID `uid` and password `pwd`). These are validated by `AuthService` and a session cookie is returned to the user. The invokation of `authenticateUser()` is indicated in figure 3 by the dependency with the stereotype «`call`».

In order to check security aspects, security requirements can be formulated in UMLsec. The fact that the invokation of `authenticateUser()` requires *secrecy* for the provided credentials is indicated in figure 3 by `EndUser`'s stereotype «`critical`» and the corresponding tagged value `{secrecy={authenticateUser()}}`. Furthermore, the returned session cookie `sc` has to be protected from manipulation, i.e. it's *integrity* is required. This is indicated by the tagged value `{integrity={sc}}`.

Information Flow Figure 4 shows the scenario in terms of the information flow between the communicating objects and is self-explanatory.

Physical Allocation The physical distribution of the communicating objects is modeled in figure 5 in the form of a UML deployment diagram. The objects of type `EndUser` and `AuthService`, respectively the components they are contained in, reside on different node instances (`clientPC` and `authServer`), which are connected by a link stereotyped «`encrypted`». As we have seen in the description of the interacting entities above, the logical communication between the components is indicated by dependencies stereotyped appropriately.

Attacker Model UMLsec defines two attacker models [5] and custom attacker models can be formulated to describe the specific abilities of an attacker. For a metasearch application in a corporate network, an attacker would have the following capabilities: Plugging his notebook into the corporate LAN (e.g., in an emtpy conference room), the attacker would have full access to the enterprise's intranet, that is, he could read, insert and delete packets and access

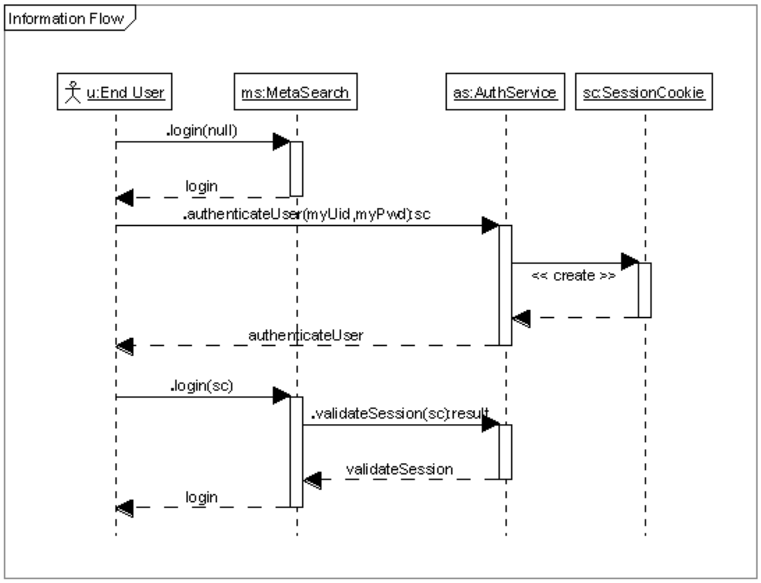


Figure 4. Information Flow

Stereotype	$\texttt{Threats}_{insider}()$
Internet	$\{\emptyset\}$
encrypted	$\{\texttt{delete}\}$
LAN	$\{\texttt{delete}, \texttt{read}, \texttt{insert}, \texttt{access}\}$
wire	$\{\emptyset\}$
smart card	$\{\emptyset\}$
POS device	$\{\emptyset\}$
issuer node	$\{\emptyset\}$

Table 1. Insider Attacker Capabilities

various network nodes. Regarding encrypted connections, he can only delete packets. These capabilities, and others, are outlined in table 1.

Static Security Analysis

In order to check the security requirements formulated in the class diagram with respect to the physical allocation and the abilities of an attacker, the various models can be integrated to a UML subsystem and a tool-supported security analysis can be processed by one of the analysis plug-ins in the UMLsec tool framework (Fig. 1b). The tool checks that the security requirements added into the UML diagrams using the stereotypes defined in the UMLsec extension are indeed supported by the design of the system. The UML subsystem description is presented in detail at [2, p. 72]. Given the described subsystem, the tool output looks as follows:

```
Name of the dependency:
  receive session cookie
Stereotype of the communication link
  of the dependency ``receive session cookie'':
  encrypted
Stereotype of the dependency:
  integrity
=> Satisfies the requirement of
  the stereotype ``secure links''.

Name of the dependency:
  send login data
Stereotype of the communication link
  of the dependency ``send login data'':
  encrypted
Stereotype of the dependency:
  secrecy
=> Satisfies the requirement of
  the stereotype ``secure links''.

=> UML model satisfies the requirement of
  the stereotype ``secure links''.
```

Thus, both the security requirements of *secrecy* and *integrity* are satisfied by the given model, provided that the encryption of the communication link between the nodes holds. We explore this in more detail in the following section.

If the tool finds that any of the security requirements that are included as stereotypes is violated, it returns, in addition to the text report, also a modified UML models where the violated stereotypes are highlighted. For instance, in a variation of the above example where the communication link over which the *send login data* dependency is implemented is not *encrypted* but a simple *Internet* link, the tool would return both the information that this violates the *secure links* security policy with regards to the *send login data* dependency, and a modification of the submitted UML model where the *secure links* and *send login data* stereotypes are highlighted.

4. Security Analysis of the Authentication Protocol

The logical connection between `clientPC` and `authServer` is based on an SSL-encrypted channel. First, the SSL handshake is executed in order to establish a secure connection, then the client (subsequently called C) sends his credentials to the authentication server (subsequently called AS) and receives a session cookie in return. An informal overview of the authentication protocol is depicted in figure 6. A detailed description of this protocol has to be omitted and can be seen in [2]. The authentication protocol was modeled as a UML sequence diagram and analyzed against the security requirements using another one of the analysis plugins in the UMLsec tool framework (Fig. 1b), which makes use of the First-Order Logic automatic theorem prover E-SETHEO [12].

Step 1: Secrecy of Pre-Master-Secret

The first steps of the protocol are as follows: C and AS exchange nonces[1] which are later used to generate shared secrets. AS sends its certificate to C. After C has validated this certificate, it generates a pre-master-secret *pms* and sends it to AS, enrcrypted with AS's public key K_{AS} extracted from the certificate. In an informal notation, these steps are described as follows:

$$\begin{array}{rcl} C \rightarrow AS & : & \texttt{ClientHello}\,(N_C(i)) \\ AS \rightarrow C & : & \texttt{ServerHello}\,(N_{AS}(j)) \\ AS \rightarrow C & : & \texttt{Certificate}\,(K_{AS}) \\ C \rightarrow AS & : & \texttt{ClientKeyExchange}\,\left(\{pms\}_{c_K}\right) \end{array}$$

with $c_K ::= \texttt{Certificate}_1$, i.e. the first argument of the message `Certificate`.

At this stage of the protocol, both C and AS possess the shared secret *pms*. It had to be shown that an adversory A cannot extract or manipulate *pms* by evesdropping the network connection or injecting his own data. Assuming that A is able to receive K_{AS}, the following conjecture had to be proved:

```
%-- Attackers Initial Knowledge --
input_formula(previous_knowledge,axiom,(
  knows(k_as))).

%-- Conjecture --
input_formula(attack,conjecture,(
  knows(pms))).
```

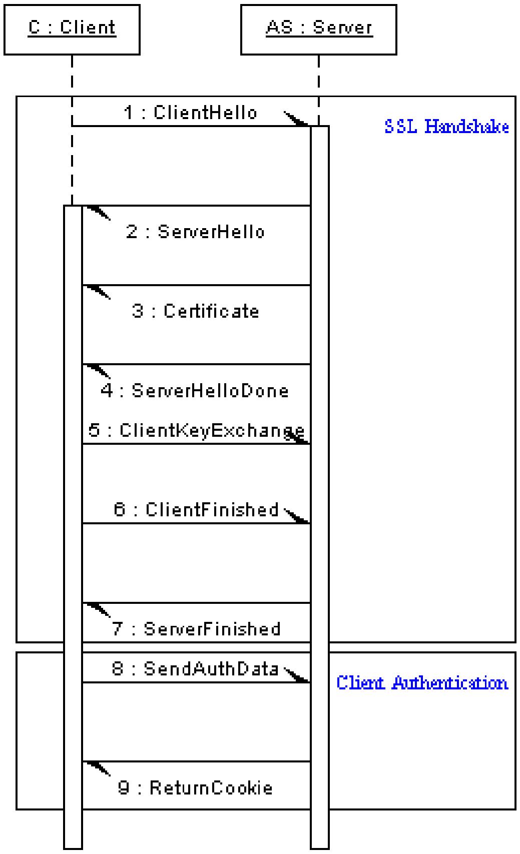


Figure 6. Informal View of the Authentication Protocol

After invoking the automated theorem prover, the UMLsec tool reported that the attack conjecture is not derivable from the axioms given the initial knowledge of A, which means that the protocol is secure with respect to the adversary model and the security requirements that were considered here.

Step 2: Realistic Adversary Model

Assuming that a realistic adversary would have its own set of cryptographic keys (K_A and K_A^{-1}), the conjecture would look as follows:

```
%-- Attackers Initial Knowledge --
input_formula(previous_knowledge,axiom,(
  knows(inv(k_a))
  & knows(k_a)
  & knows(k_as)
  )).
%-- Conjecture --
input_formula(attack,conjecture,(
  knows(pms))).
```

[1] A nonce is a freshly generated random value that should only be used once for security-relevant purposes.

Now the UMLsec tool reported that the threat conjecture is derivable from the system assumptions, which means that A was able to extract *pms*. This is possible in the following scenario: When AS and C exchange K_{AS}, A could replace K_{AS} with his own key K_A. Thus, C encrypts *pms* with K_A, which can then be decrypted by A. This scenario is depicted in the following message flow diagram:

$$C \quad \xleftarrow{K_A} \quad A \quad \xleftarrow{K_{AS}} \quad AS$$

$$C \quad \xrightarrow{\{pms\}_{K_A}} \quad A \quad \xrightarrow{\{pms\}_{K_{AS}}} \quad AS$$

This scenario arises because of an inadequate model of the protocol: the public key of the client has to be protected from manipulation, and this was not captured in the model. In order to model the protocol flow correctly, the entire certificate of AS has to be modeled in the message `Certificate`.

Step 3: Model of the Certificate

In order to model the certificate correctly, K_{AS} is concatenated with AS (the name of *AS*) and signed by an independent Certificate Authority *CA*. The modified message now looks like this:

$$AS \rightarrow C \quad : \quad \texttt{Certificate}\left(Sign_{K_{CA}^{-1}}\left(K_{AS} :: AS\right)\right)$$

C now sends *pms* only if the condition

$$[\textbf{snd}\,(Ext_{K_{CA}}\,(c_K)) = AS]$$

with $c_K ::= \texttt{Certificate}_1$ holds. In this case, the following message is sent:

$$C \rightarrow AS \quad : \quad \texttt{ClientKeyExchange}\left(\{pms\}_{\textbf{fst}\left(Ext_{K_{CA}}(c_K)\right)}\right)$$

The secrecy of *pms* now has to be checked once more:

```
%-- Attackers Initial Knowledge --
input_formula(previous_knowledge,axiom,(
  knows(k_ca)
  & knows(inv(k_a))
  & knows(k_a)
  & knows(k_as)
  )).
%-- Conjecture --
input_formula(attack,conjecture,(
  knows(pms) )).
```

The tool now reported that the threat conjecture cannot be derived from the logical assumptions on the system, which means that no attack of the kind under consideration could be found and that *pms* is secure regarding this kind of attacks.

Based on *pms*, *C* and *AS* can now independently compute a symmetric session key. This session key consists mainly of a hash over the concatenation of a so called *master secret* with the before exchanged nonces. The master secret itself consists of a hash over the concatenation of *pms* with those nonces. To improve readability, the computed session keys can be formalized as follows:

$$\begin{aligned} SK_C \quad &::= \quad hash\,(pms :: N_C(i) :: c_{sn}) \\ SK_{AS} \quad &::= \quad hash\left(Dec_{K_{AS}^{-1}}\,(s_{pms}) :: s_{cn} :: N_{AS}(j)\right) \end{aligned}$$

where $c_{sn} ::= \texttt{ServerHello}_1$, $s_{cn} ::= \texttt{ClientHello}_1$ and $s_{pms} ::= \texttt{ClientKeyExchange}_1$.

The messages `ClientFinished` and `ServerFinished` have been omitted, as they do not carry security critical information.

Step 4: Secure Exchange of the User Password

C now sends *AS* his authentication credentials encrypted with the symmetric session key SK_C:

$$C \rightarrow AS \quad : \quad \texttt{SendAuthData}\left(\{uid :: pwd\}_{SK_C}\right)$$

where uid is the user-id and pwd is the password. The secrecy of pwd can now be proved as follows:

```
%-- Attackers Initial Knowledge --
input_formula(previous_knowledge,axiom,(
  knows(k_ca)
  & knows(inv(k_a))
  & knows(k_a)
  & knows(k_as)
  )).
%-- Conjecture --
input_formula(attack,conjecture,(
  knows(pwd) )).
```

Again, the tool reported that the attack conjecture is not derivable from the axioms, *pwd* is secure with respect to the adversary model and the security requirements that were considered here.

Step 5: Secure Exchange of the Session Cookie

Once *AS* has validated *C*'s authentication information, it generates a session cookie $cookie$ and sends it, encrypted with its symmetric session key SK_{AS}, to *C*:

$$AS \rightarrow C \quad : \quad \texttt{ReturnCookie}\left(\{cookie\}_{SK_{AS}}\right)$$

We can now test if *A* can read *cookie*:

```
%-- Attackers Initial Knowledge --
input_formula(previous_knowledge,axiom,(
  knows(k_ca)
  & knows(inv(k_a))
  & knows(k_a)
  & knows(k_as)
  )).
%-- Conjecture --
input_formula(attack,conjecture,(
  knows(cookie) )).
```

Here, the UMLsec tool reported that the attack conjecture can be derived from the system assumptions, which indicates that *A* was able to read $cookie$, that is, it is not secure! The reason is the following: as *A* knows the public key K_{AS} of *AS*, it could catch the message `ClientKeyExchange` containing *pms* and send a modified pre master secret pms' to *AS*. *AS* now computes its symmetric session key based on pms' and sends the session cookie $cookie$ encrypted with this session key. As *A* and *AS* both compute the same session keys based on pms', *A* is able to decrypt the message `ReturnCookie`. This situation is depicted in message flow notation as follows:

$$\begin{tikzcd}
C \arrow[r, "\{pms\}_{K_{AS}}"] & A \arrow[r, "\{pms'\}_{K_{AS}}"] & AS \\
 & \vdots & \\
C & A \arrow[l, "\{cookie\}_{SK'_{AS}}"'] & AS \arrow[l, "\{cookie\}_{SK'_{AS}}"']
\end{tikzcd}$$

where SK'_{AS} is the session key computed by *AS* based on pms'. To avoid this situation, it has to be guaranteed that pms cannot be modified by A.

So, the model of the protocol has to guarantee that the message `ReturnCookie` is only sent from *AS* to *C* in case the pre master secret pms was not modified. This is achieved in the SSL specification by two *Finished*-messages which are exchanged at the end of the SSL handshake and contain a message authentication code (MAC) over all prior exchanged messages. These two messages are formalized as follows, where $handshake\_messages$ stands for the prior messages:

$$C \rightarrow AS \;:\; \texttt{ClientFinished}\left(Mac_{SK_C}(handshake\_messages)\right)$$
$$AS \rightarrow C \;:\; \texttt{ServerFinished}\left(Mac_{SK_{AS}}(handshake\_messages)\right)$$

The exchange of $cookie$ is guarded by the following condition connected to message `ReturnCookie`:

$$[Dec_{SK_{AS}}(c_H) = handshake\_messages]$$

where c_H ::= $\texttt{ClientFinished}_1$. Now, the secrecy of $cookie$ has to be proved once more:

```
%-- Attackers Initial Knowledge --
input_formula(previous_knowledge,axiom,(
  knows(k_ca)
  & knows(inv(k_a))
  & knows(k_a)
  & knows(k_as)
  )).
%-- Conjecture --
input_formula(attack,conjecture,(
  knows(cookie) )).
```

Finally, the UMLsec tool reported that the attack conjecture is not derivable from the axioms and therefore $cookie$ is secure with respect to the adversary model and the security requirements that were considered here.

The complete sequence diagram is depicted in Fig. 7.

5. Lessons Learned

In this section, we provide a discussion of the lessons learned from this application experience at the hand of some guiding questions.

Are there ways in which the application of UMLsec did not go as expected? Generally, the application of UMLsec worked as expected. One has to note that analyzing sophisticated security mechanisms such as cryptographic protocols turned out to be less easy than applying simple consistency checks (such as the secure dependency check). However, most developers will not design their own crypto protocols but instead just have to make sure that they use existing protocols correctly.

Did the method have to be changed or adapted to work properly, and if so, in what way? Did UMLsec rules have to be changed to fit the application ? The method, and in particular the UMLsec rules, worked properly without having to be changed or adapted.

Did the method yield interesting or unexpected results? The method showed that the system under consideration is indeed secure with respect to the security requirements and adversary model that were considered, after all necessary details had been incorporated in the models in a step-by-step approach. This result is certainly interesting giving the high number of insecure systems in practical use.

Did it not pick up issues that you would have hoped it would or should? To demonstrate that the method does pick up the security design flaws as expected, we applied it incrementally to a model of the protocol built up in a step-by-step way. As expected, we could demonstrate that all security features in the protocol were indeed needed to provide the mandated level of security.

How did its use differ from previous uses? Are all applications the same? To our knowledge, this was the first application of UMLsec to a general business application (more specifically, an information management system). It differs from previous applications in so far as the general environment of the business application had to be taken into account to properly capture the threat scenario from a realistic adversary. This could be done rather easily using the adversary definition mechanism built into UMLsec making use of so-called “threat sets”.

Can you say anything specific about the security of the application/system now that you have done the modelling? Using the UMLsec approach, we were able to precisely demonstrate whether certain security requirements central to the application are actually correctly enforced by the application. It turned out that this is actually the case.

How can you be sure you have applied the method correctly or even optimally? Are there other ways in which you could have applied it? By applying a step-by-step approach, we were able to demonstrate that for an insufficient model of the system, the tools actually found the relevant security weaknesses. This increases our trust in the tool that, if there were any further security design weaknesses of the kind the tool is built to detect, it would have found them. We tried to apply the UMLsec method optimally in the sense that we focussed on the security-critical core of the information management system. Since the identification of this part was done informally, there always exist the possibility that security weaknesses may have gone undetected for this reason. However, we believe that this approach is the most cost-effective one in a practical application in industry (although there currently does not seem to exist a practical approach to compute an optimal Return-on-Security-Investement, in particular not when applied to software development process decisions).

General Discussion. Overall, the use of UMLsec was generally appropriate for this case study. We were able to start the security analysis on a rather abstract level by proving static security requirements, as well as to conduct the security analysis of the authentication protocol on a very detailed level. However, there remain areas for future improvement of the UMLsec method. While the definition of the static security requirements is quite intuitive for the developer, the modeling of the security aspects of the authentication protocol turned out to be rather complex for a developer who has only a broad experience with security. Future improvements on the usability of the method would be useful.

By using UMLsec in the case study, the conformance of the system models to the security requirements was demonstrated. However, this has no implication on the real system, as long as the application code is not generated directly from the model and there is no assurance that the generation process itself is correct. Therefore, conclusions from the model-based security analysis have to be drawn carefully, but it is clear that the analysis results provide useful input into the development process. Research on linking model-level security analysis to the implementation level is currently under way [8].

Although this case-study was not aimed at assessing the usefulness of model-based software development techniques in general, it became obvious that such techniques do come with an added overhead with respect to training of the developers and with respect to development effort and time. It can therefore be expected that uptake of model-based software development in industry be fastest for application domains that involve highly sophisticated and critical requirements, such as security-critical systems. In these areas, the effort of using model-based design and analysis techniques (such as UMLsec) is comparable to (or even better than) the effort needed with traditional assurance approaches that deliver a comparable level of trustworthiness of the system under development, such as traditional formal methods (although again, a controlled comparative study on this observation was not part of the scope of the current case-study but would be very interesting future work). Also, the possibility to perform sophisticated automated security analysis routines on the models that are constructed provides additional incentive. This is in addition to the fact that using model-based techniques lead to a more stringent development process, and the fact that high security assurance levels (e.g. according to the Common Criteria [3]) anyway require developers to create extensive and precise (on the highest levels even semi-formal or formal) documentation of their systems.

6. Conclusion

This paper presented a field report on the deployment of the UMLsec method in an industrial context. A model-based security analysis was conducted on a search engine in the intranet of a major German car manufacturer. The focus was on the application’s single-sign-on-mechanism.

Using the UMLsec notation, the developer was able to annotate his models with information regarding the security critical aspects of the system in a concise and clear way. Employing the UML profile of UMLsec, developers familiar with the extension mechanisms of the UML should have no problem to learn UMLsec quickly.

Furthermore, by embedding the security analysis directly into the development process, a better understanding and clearer communication of these issues is made possible.

Acknowledgements We would like to thank Guido Wimmel for carefully reading draft versions of this paper and for providing valuable feedback.

References

[1] A. Apvrille and M. Pourzandi. Secure software development by example. *IEEE Security & Privacy*, 3(4):10–17, 2005.

[2] B. Best. Model-based Security Analysis of Industrial Information Management Systems at the Example of a Meta-Search-Machine in the Intranet of the BMW Group. Technical report, TU Munich, 2006.

[3] Common Criteria. `http://csec.nist.gov/cc/`, 2001.

[4] J. Grünbauer, H. Hollmann, J. Jürjens, and G. Wimmel. Modelling and verification of layered security-protocols: A bank application. In *SAFECOMP 2003*, Lecture Notes in Computer Science. Springer-Verlag, 2003.

[5] J. Jürjens. *Secure Systems Development with UML*. Springer-Verlag, 2004.

[6] J. Jürjens. Sound methods and effective tools for model-based security engineering with UML. In *27th Int. Conf. on Softw. Engineering*. IEEE Computer Society, 2005.

[7] J. Jürjens. Security analysis of crypto-based Java programs using automated theorem provers. In *21st IEEE/ACM Int. Conf. Autom. Softw. Eng.*, 2006.

[8] J. Jürjens. Security analysis of crypto-based Java programs using automated theorem provers. In S. Easterbrook and S. Uchitel, editors, *21st IEEE/ACM International Conference on Automated Software Engineering (ASE 2006)*. ACM, 2006.

[9] J. Jürjens and J. Fox. Tools for model-based security engineering. In *28th International Conference on Software Engineering (ICSE 2006)*. ACM, 2006.

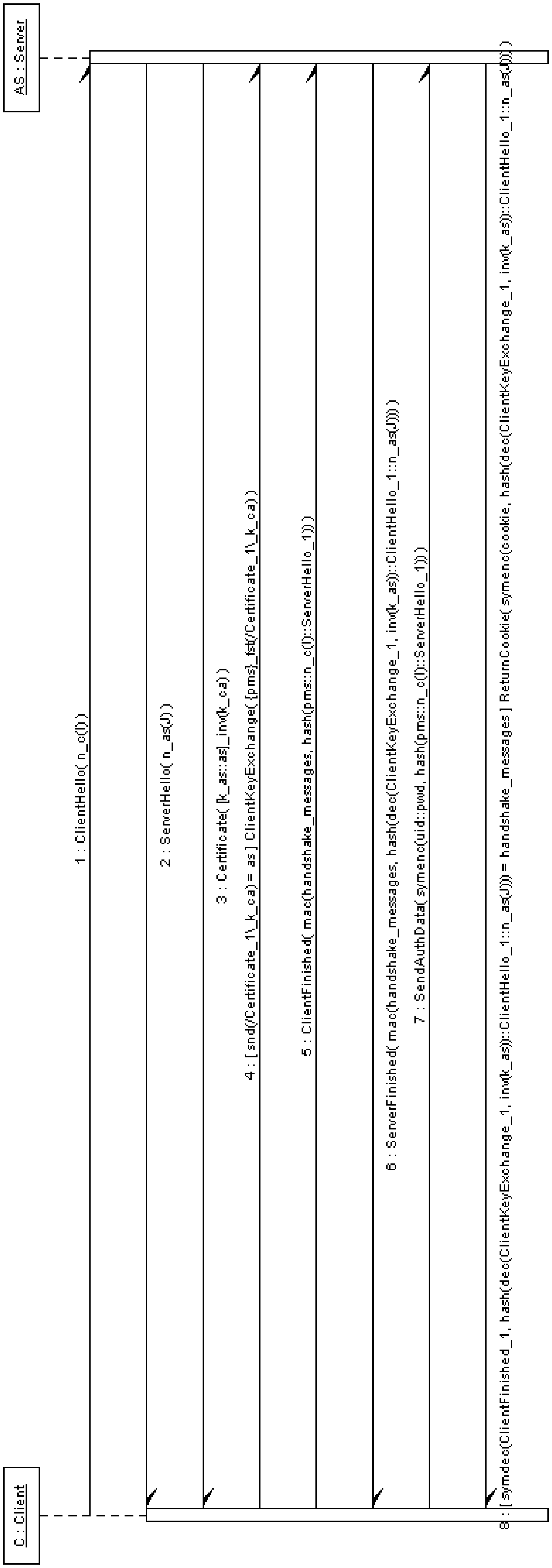
[10] D. Perry, A. Porter, and L. Votta. Empirical studies of software engineering: a roadmap. In *ICSE - Future of SE Track*, pages 345–355, 2000.

[11] J. Schalken. Research methods for the empirical assessment of software processes. In *The 12th Doctoral Consortium at CAiSE 05*, 2005.

[12] G. Stenz and A. Wolf. E-SETHEO: An automated^3 theorem prover. In R. Dyckhoff, editor, *Automated Reasoning with Analytic Tableaux and Related Methods (TABLEAUX 2000)*, volume 1847 of *Lecture Notes in Computer Science*, pages 436–440. Springer-Verlag, 2000.

[13] UMLsec group. Security analysis tool, 2004. `http://www.umlsec.org`.

[14] M. Vetterling, G. Wimmel, and A. Wisspeintner. Secure systems development based on the Common Criteria. In *10th International Symposium on the Foundations of Software Engineering (FSE-10)*, pages 129–138. ACM, 2002.



AS : Server
C : Client
1 : ClientHello(n_c(J))
2 : ServerHello(n_as(J))
3 : Certificate([k_as::as]_inv(k_ca))
4 : [snd(/(Certificate_1_k_ca) = as] ClientKeyExchange({pms}_fst(/(Certificate_1_k_ca))
5 : ClientFinished(mac(handshake_messages, hash(pms::n_c(J)::ServerHello_1)))
6 : ServerFinished(mac(handshake_messages, hash(dec(ClientKeyExchange_1, inv(k_as))::ClientHello_1::n_as(J))))
7 : SendAuthData(symenc(uid::pwd, hash(pms::n_c(J)::ServerHello_1)))
8 : [symdec(ClientFinished_1, hash(dec(ClientKeyExchange_1, inv(k_as))::ClientHello_1::n_as(J))) = handshake_messages] ReturnCookie(symenc(cookie, hash(dec(ClientKeyExchange_1, inv(k_as))::ClientHello_1::n_as(J))))

Figure 7. Complete model of the authentication protocol