Identification of Vulnerabilities in Web Services using Model-based Security

Sebastian Höhn
sebastian.hoehn@iig.uni-freiburg.de
Albert-Ludwig University
Institute of Computer Science and Social Studies
Freiburg, Germany

Jan Jürjens
J.Jurjens@open.ac.uk
Computing Department
Open University

Lutz Lowis
lutz.lowis@iig.uni-freiburg.de
Albert-Ludwig University
Institute of Computer Science and Social Studies
Freiburg, Germany

Rafael Accorsi
rafael.accorsi@iig.uni-freiburg.de
Albert-Ludwig University
Institute of Computer Science and Social Studies
Freiburg, Germany

Abstract

In a service-oriented architecture, business processes are executed as composition of services, which can suffer from vulnerabilities. These vulnerabilities in services and the underlying software applications put at risk computer systems in general and business processes in particular. Current vulnerability analysis approaches involve several manual tasks and, hence, are error-prone and costly. Service-oriented architectures impose additional analysis complexity as they provide much flexibility and frequent changes within orchestrated processes and services. Therefore, it is inevitable to provide tools and mechanisms that enable efficient and effective management of vulnerabilities within these complex systems. Model-based security engineering is a promising approach that can help to fill the gap between vulnerabilities on the one hand, and concrete protection mechanisms on the other.

We present an approach that integrates model-based engineering and vulnerability analysis in order to cope with the security challenges of a service-oriented architecture.

Introduction

Information systems consist of a plethora of different applications, services and components. The complex interplay between these system parts is one of the main challenges for the establishment of reliable and secure service oriented architectures (SOA). Among the prominent requirements for enterprise information systems is the ability to react to changes quickly and flexibly. To this end, a SOA is deployed in many different application scenarios. It allows the orchestration of services and the implementation of complex business processes without implementing the basic functions over and over again.

Security concepts for SOA heavily rely on model-based technologies. This is due to two prominent reasons: (1) model-based mechanisms work reliably and fast even in complex industrial settings, and (2) SOA itself is a model-based architecture. The deployment and the execution of business processes in a SOA are based on executable business process models mostly written in BPEL. The description of atomic services and their composition to higher-order services is also done in BPEL-Models, together with a WSDL description of the implemented interfaces.

To this end, we propose the integration of model-based security mechanisms for SOA. Current approaches (as explained in the next chapter) neglect the fact that vulnerabilities are major source for security incidents. In classical systems, vulnerability analysis and integration of appropriate counter-mechanisms is a mainly manual task. This is possible because these systems are quite static: they are deployed once and used for longer period in time. In a SOA, systems are composed and re-composed frequently and it becomes infeasible to manually interact with specific instances of business processes or high-order services. For example, they might be part of a complex orchestration. While it might seem a strong assumption that processes and services are orchestrated for unique tasks, systems exist that allow for dynamic integration of additional steps into existing processes (Reichert et al., 2006): by integrating individually required steps, a unique process arises that is executed exactly once.

These scenarios clearly show that security information and vulnerability information must be prepared for automated processing. If users can integrate additional process steps into existing business processes on the fly, it is inevitable to automatically evaluate the security implications. Several security properties of the resulting processes can be evaluated automatically. The following section will provide a motivation for and an overview of these mechanisms. Afterwards, we present a model-based extension for UMLsec that allows for the automated evaluation of vulnerabilities and their effects in a SOA.

Model-based Security Analysis

Challenges for Computer Security

Attacks against computer systems, on which the infrastructures of modern society and modern economies rely, cause substantial financial damage. Due to the increasing interconnection of systems, such attacks can be waged anonymously and from a safe distance. Thus networked computers need to be secure. The high-quality development of security-critical systems is difficult. Still, many systems are developed, deployed, and used over years that contain significant security weaknesses. Causes: While tracing requirements during software development is difficult enough, enforcing security requirements is intrinsically subtle, because one has to take into account the interaction of the system with motivated adversaries that act independently. Thus security mechanisms, such as security protocols, are notoriously hard to design correctly, even for experts. Also, a system is only as secure as its weakest part or aspect. Security is compromised most often not by breaking dedicated mechanisms such as encryption or security protocols, but by exploiting weaknesses in the way they are being used (Anderson & Long, 2001). Thus it is not enough to ensure correct functioning of security mechanisms used. They cannot be “blindly” inserted into a security-critical system, but the overall system development must take security aspects into account in a coherent way (Saltzer & Schroeder, 1975). In fact, according to (Schneider, 1998), 85% of Computer Emergency Response Team (CERT) security advisories could not have been prevented just by making use of cryptography. Building trustworthy components does not suffice, since the interconnections and interactions of components play a significant role in trustworthiness (Schneider, 1998).

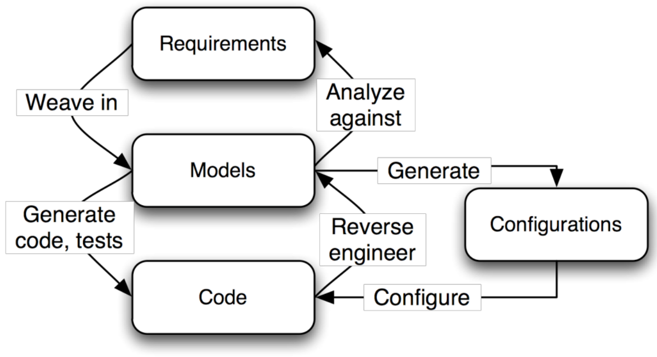


Figure 1: Model-based Security Engineering

State of the Art in Model-Based Security

In practice, the traditional strategy for security assurance has been “penetrate and patch”: It has been accepted that deployed systems contain vulnerabilities. Whenever a penetration of the system is noticed and the exploited weakness can be identified, the vulnerability is removed. Sometimes this is supported by employing friendly teams trained in penetrating computer systems, the so-called “tiger teams”. However, this approach is not ideal: Each penetration using a new vulnerability may already have caused significant damage, before the vulnerability can be removed. It would thus be preferable to consider security aspects more seriously in earlier phases of the system life-cycle, before a system is deployed, or even implemented, because late correction of requirements errors costs up to 200 times as much as early correction (Boehm, 1981). Also, security concerns must be taken into account during every phase of software development, from requirements engineering to design, implementation, testing, and deployment. Academic approaches trying to improve security during development must be *tightly integrated* with software development approaches already used in industry (Devanbu & Stubblebine, 2000). Some other challenges for using sound engineering methods for secure systems development exist. For example, the boundaries of specified components with the rest of the system need to be carefully examined, for example with respect to implicit assumptions on the system context (Gollmann, 2000). Lastly, a more technical issue is that formalized security properties are not in all approaches preserved by refinement. Since an implementation is necessarily a refinement of its specification, an implementation of a secure specification may, in such a situation, not be secure, which is clearly undesirable. A truly secure software engineering approach thus needs to take both dimensions of the problem into account:

- it needs to integrate the different system lifecycle phases,
- it also needs to take into account the different architectural levels of abstraction of a security-critical system in a demonstrably sound, trustworthy, and cohesive way.

The approach in the second part of this chapter presents how we can cope with possible vulnerabilities arising from the refinement of the model into an actual application.

Approaching Model-Based Security

In this section, we give an overview on some approaches for secure software engineering, with an emphasis on model-based development using UML. Of course, there are also approaches for secure software engineering outside of or predating model-based development with UML (cf. for example (Anderson & Long, 2001; Devanbu & Stubblebine, 2000; Eckert & Marek, 1997; Saltzer & Schroeder, 1975; Schneider, 1998) for some examples and overviews), but because of space restrictions, we cannot consider those in detail here.

Model-based Development

Generally, when using model-based development (cf. Figure 1), the idea is that 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: can first check that the systems fulfils 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 the security 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. One then extracts models from the source code, which can then again be analyzed against the security requirements. Using model-based development, one can also incorporate the configuration data (such as user permissions) in the analysis, which is very important for security but often neglected.

For example, in the Model-based Security Engineering (MBSE) approach based on the UML extension UMLsec (Jürjens, 2002; Jürjens, 2005a; Jürjens, 2005b), 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. 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. 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 assumptions on the physical level of the system, security requirements related to the secure handling and communication of data, and security policies that system parts are supposed to obey. The semantics for the fragment of UML used for UMLsec is defined in (Jürjens, 2005a) using so-called UML Machines, which is a kind of state machine with input/output interfaces similar to Broy's Focus model, whose behaviour can be specified in a notation similar to that of Abstract State Machines (ASMs), and which is equipped with UML-type communication mechanisms. On this basis, important security requirements such as secrecy, integrity, authenticity, and secure information flow are defined.

After an early paper on the UML extension UMLsec for secure software development (Jürjens, 2001b), a number of approaches have been developed each targeted at certain facets of model-based development of security-critical systems using UML, several of them initially presented at the workshop series CSDUML (Critical Systems Development using Modeling Languages). A general development process in this context was proposed in (Breu, R., Burger, K., Hafner, M., Jürjens, J., Popp, G., Wimmel, G. & Lotz, V., 2003; Popp, G., Jürjens, J., Wimmel, G. & Breu, R., 2003).

Security requirements modelling: (Fernandez & Hawkins, 1997) proposes a method determining role-based access rights. Use cases are extended with rights specifications and the rights of a role are derived from the use cases. The method thus enforces the design principle of least privilege. (Crook, Ince, Lin & Nuseibeh, 2002; Haley, Laney, Moffett & Nuseibeh, 2008) formulates a vision for the requirements engineering community towards providing a "bridge between the well-ordered world of the software project informed by conventional requirements and the unexpected world of anti-requirements associated with the malicious user". (Giorgini, Massacci, Mylopoulos & Zannone, 2005; Massacci, Mylopoulos & Zannone, 2007; Mylopoulos, Giorgini & Massacci, 2003) proposes an extension of the i*/Tropos requirements engineering framework to deal with security requirements. The Tropos Requirements Engineering methodology is also extended to cover security aspects in (Manson, Mouratidis & Giorgini, 2003). (Sindre & Opdahl, 2005) presents an approach to eliciting security requirements using use cases which extends traditional use cases to also cover misuse. (Fox, Mouratidis & Jürjens, 2006) uses a combination of UMLsec and Tropos to get a transition from the security requirements to the design phase. (Whittle, Wijesekera & Hartong, 2008) presents an executable misuse case modelling language, which allows modellers to specify misuse case scenarios in a formal yet intuitive way and to execute the misuse case model in tandem with a corresponding use case model. (Yskout, Scandariato, Win & Joosen, 2008) presents an approach for the transformation of security requirements to software architectures. (Arenas, Aziz, Bicarregui, Matthews & Yang, 2008) discuss the use of requirements-engineering techniques in capturing security requirements for a Grid-based operating system. (Yu & Elahi, 2008) examines how conceptual modelling can provide support for analyzing security trade-offs, using an extension to the i* framework. (Flechais, Mascolo & Sasse, 2007) presents an approach that integrates security and usability into the requirements and design process, based on a development process and a UML meta-model of the definition and the reasoning over the system's assets.

Security patterns: (Yoshioka, Honiden & Finkelstein, 2004) proposes a UML based method that enables developers to specify several candidate system behaviours that satisfy the security requirements, using patterns, and shows an application of the method to a real implemented system, the Environmentally Conscious Product (ECP) design support system. A methodology to build secure systems using patterns is presented in (Fernandez, Larrondo-Petrie, Sorgente & VanHilst, 2006). A main idea in the proposed methodology is that security principles should be applied at every stage of the software lifecycle and that each stage can be tested for compliance with those principles. Another basic idea is the use of patterns at each stage. A pattern is an encapsulated solution to a recurrent problem and their use can improve the reusability and quality of software. (Rosado, Fernandez-Medina, Piattini & Gutierrez, 2006) compares several security patterns to be used when dealing with application security. The SERENITY approach is based on the notion of Security and Dependability Patterns (Maña et al., 2007). They include a functional description of the proposed security solution, a semantic description of the security requirements addressed by it, and descriptions of the assumptions on the context in which the pattern can be used.

Automated security verification: So far only few of the UML based approaches for secure software development come with automated tools to formally verify the UML design for the relevant security requirements. One of these is again the UMLsec approach. The UMLsec tool-support can 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 (Jürjens, 2005b; Jürjens & Shabalin, 2007; Shabalin & Jürjens, 2004; UMLsec group, 2009). They 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. Since the analysis that is performed is too sophisticated to be done manually, it is also valuable to security experts. 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. Other approaches for verifying UML models for security properties emerge. For example, (Egea, Basin, Clavel & Doser, 2007) explains an approach in which queries about properties of an RBAC policy model are expressed as formulas in UML's Object Constraint Language and evaluated over the metamodel of the security-design language, based on the rewriting logic Maude. Also, (Siveroni, Zisman & Spanoudakis, 2008) presents a tool for verifying UML class and state machine diagrams against linear temporal logic formulas using Spin, which is planned to be applied to security properties.

Model construction and development: Having a formally based design notation allows one to precisely formulate and investigate non-trivial questions that need to be solved to enable trustworthy secure software development. For example, to support stepwise development, it has been shown that within UMLsec, secrecy, integrity, authenticity, and secure information flow are preserved under refinement and the composition of system components (under suitable assumptions) (Jürjens, 2005a). Similarly, it has been shown that layering of security services (such as layered security protocols) is sound, again only under certain assumptions. The same applies to the application of security design patterns, or the use of aspect-oriented modeling techniques. Related approaches have been reported in (Santen, 2006; Santen, Heisel & Pfitzmann, 2002).To support the Security and Dependability Patterns used in the project, the SERENITY approach also provides Security and Dependability Schemes which allow the users to combine existing security solutions to more complex ones (Maña et al., 2007). Automated tools for classification, selection, and composition of security patterns support this.

Aspect-Oriented Security Modeling: (Houmb, Georg, France, Bieman & Jürjens, 2005; Ray, France, Li & Georg, 2004) propose to use aspect-oriented modeling for addressing access control concerns. Functionality that addresses a pervasive access control concern is defined in an aspect. The remaining functionality is specified in a so-called primary model. Composing access control aspects with a primary model then gives a system model that addresses access control concerns. Model-based Security Risk Assessment: (Dimitrakos et al., 2002) uses UML for the risk assessment of an e-commerce system within the CORAS framework for model-based risk assessment. This framework is characterized by an integration of aspects from partly complementary risk assessment methods. It incorporates guidelines and methodology for the use of UML to support and direct the risk assessment methodology as well as a risk management process based on standards such as AS/NZS 4360 and ISO/IEC 27002. It uses a risk documentation framework based on RM-ODP together with an integrated risk management and system development process based on UP and offers a platform for tool inclusion based on XML. In another approach (Baldwin, Beres, Shiu & Kearney, 2006; Kearney & Brügger, 2007) use UML for risk-driven security analysis that focuses on the assessment of risk and analysis of requirements for operational risk management.

Secure business processes and Service-oriented architectures: A business process driven approach to security engineering using UML is presented in (Maña, Montenegro, Rudolph & Vivas, 2003). The idea is to use UML models in an approach centered on business processes to develop secure systems. A model-based security engineering approach for developing service-oriented architectures is proposed in (Deubler, Grünbauer, Jürjens & Wimmel, 2004). The approach is applied on a standard for service-oriented architectures from the Automotive domain (OSGi). Access control policies: (Basin, Doser & Lodderstedt, 2006; Wolff, Brucker & Doser, 2006) show how UML can be used to specify access control in an application and how one can then generate access control mechanisms from the specifications. The approach is based on role-based access control and gives additional support for specifying authorization constraints. (Koch & Parisi-Presicce, 2006; Pauls, Kolarczyk, Koch & Löhr, 2006) demonstrate how to deal with access control policies in UML. The specification of access control policies is integrated into UML. A graph-based formal semantics for the UML access control specification permits one to reason about the coherence of the access control specification. An aspect-oriented approach to specifying access control in UML is presented in (Zhang, Baumeister, Koch & Knapp, 2005). (Méry & Merz, 2007) presents an approach for the specification and refinement of access control rules, including proof rules for verifying that an access control policy is correctly implemented in a system, and preservation of access control by refinement of event systems. (Hafner, Memon & Alam, 2008) presents usage scenarios for access control in contemporary healthcare scenarios and shows how to unify them in a single security policy model. Based on this model, the SECTET (Alam, Breu & Hafner, 2007) framework for Model Driven Security is then specialized towards a domain-specific approach for healthcare scenarios, including the modelling of access control policies, a target architecture for their enforcement, and model-to-code transformations. (Agreiter, Alam, Hafner, Seifert & Zhang, 2007) extends the SECTET approach to take into account operating system level and application level security mechanisms to realize security-critical application and services for healthcare scenarios.

Health information systems: There have been several approaches using UML for security aspects in developing health-care systems. (Blobel, Nordberg, Davis & Pharow, 2006) presents an approach based on formal models where security services can be integrated into advanced systems architectures enabling semantic interoperability in the context of trustworthiness of communication and co-operation to support application security challenges such as privilege management and access control. The approach covers domains, service delegation, claims control, policies, roles, authorisations, and access control. (Mathe et al., 2007) presents an approach based on model-based design techniques and high-level modelling abstractions which provides a framework to rapidly develop, simulate, and deploy clinical information system (CIS) prototypes. It includes a graphical design environment for developing formal system models and generating executable code for deployment.

Secure database design: An approach to designing the content of a security critical database uses the Object Constraint Language (OCL) which is an optional part of the Unified Modeling Language (UML). More specifically, (Piattini & Fernández-Medina, 2004) presents the Object Security Constraint Language V.2. (OSCL2), which is based in OCL. This OCL extension can be used to incorporate security information and constraints in a Platform Independent Model (PIM) given as a UML class model. The information from the PIM is then translated into a Platform Specific Model (PSM) given as a multilevel relational model. This can then be implemented in a particular Database

Management System (DBMS), such as Oracle9i Label Security. These transformations can be done automatically or semi-automatically using OSCL2 compilers.

Smart-card based applications: (Haneberg, Reif & Stenzel, 2002; Moebius, Haneberg, Reif & Schellhorn, 2007) present a method for the development of secure smartcard applications which includes UML models enriched by algebraic specifications, and dynamic logic for JavaCard verification. The approach is implemented in the KIV specification and verification system.

Secure information flow: (Hultin & Heldal, 2003) provides support for the use of UML with secrecy annotations so that the code produced from the UML models can be be validated by the Java information flow (Jif) language-based checker. (Seehusen & Stølen, 2006) provides an approach which can analyze secure information properties in UML sequence diagrams.

Model-based Security Engineering with UMLsec

We now explain the idea of model-based security engineering in more technical detail at the hand of one of the approaches mentioned in the previous section, namely the UMLsec approach. We describe a simplified fragment of UMLsec. For more details and a formal semantics cf. (Jürjens, 2002; Jürjens, 2005a; Jürjens, 2005b).

UML consists of diagram types describing different views on a system.

Use case diagrams describe typical interactions between a user and a computer system (or between different components of a computer system).

Activity diagrams can be used e.g. to model workflow and to explain use cases in more detail.

Class diagrams define the static structure of the system: classes with attributes and operations/signals and relationships between classes.

Interaction diagrams, which may be sequence diagrams or collaboration diagrams, describe interaction between objects via message exchange. Here we consider sequence diagrams; collaboration diagrams are very similar.

Statechart diagrams give the dynamic behavior of an individual object: events may cause state in change or actions.

Package diagrams can be used to group parts of a system together into higher-level units.

Deployment diagrams describe the underlying physical layer; we use them to ensure that the physical layer meets security requirements on communication.

UML offers rich extension mechanisms in form of labels. These can be either stereotypes (written in double angle brackets such as <<*stereotype*>>) or tag-value pairs (written in curly brackets such as {tag, value}). Using *profiles* or *prefaces* one can give a specific meaning to model elements marked with these labels. Here we give the extension UMLsec of UML making use of such labels to express security requirements.

We stress that the aspects not mentioned here (such as association and generalization in the case of class diagrams) can and should be used in the context of UMLsec; they do not appear in our presentation simply because they are not needed to specify the considered security properties.

Since we are using a formal fragment of UML, we may reason formally, showing e.g. that a given system is as secure as certain components of it. This way, we can reduce security of the system to the security of the employed security mechanisms (such as security protocols). For example, we can exhibit the conditions under which protocols can be used securely in the system context. Work on refining security properties aids in verification applied within the design process.

Capturing Security Requirements with Use Case Diagrams

Use case diagrams describe typical interactions between a user and a computer system (or between different components of a computer system). We use them to capture security requirements.

To start with our example, Figure 2 gives the use case diagram describing the situation to be achieved: a customer buys a good from a business. The semantics of the stereotype <<*fair exchange*>> is, intuitively, that the actions "buys good" and "sells good" should be linked in the sense that if one of the two is executed then eventually the other one will be (where these actions are specified on the next more detailed level of specification).

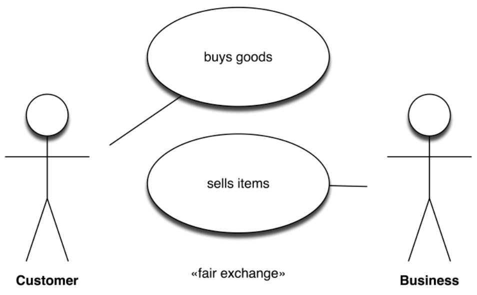


Figure 2: Exemplary use case diagram: "fair exchange"

Secure Business Processes with Activity Diagrams

Activity diagrams are especially useful to model workflow and to explain use cases in more detail. Following our example, Figure 3 explains the use case in Figure 2 in more detail. To demonstrate the connection between this diagram and the one in Figure 2, we give two possible diagrams. Both are separated in two *swim-lanes* describing activities of different parts of a system (here Customer and Business). Round boxes describe actions (such as Request Good) and rectangular boxes describe the object flow (such as Order[filled]). Horizontal bars (*synchronisation bars*) describe a required synchronisation between two different strands of activity. A diamond describes the merging of two strands of activities. The start state is marked by a full circle, the final state by a small full circle within a circle. In each diagram, two tag-value pairs {fair exchange, buys goods} and {fair exchange, sells goods} are used to mark certain actions. Now any such diagram fulfills the security requirement "fair exchange", given in the diagram in Figure 2, if for both actions marked with these tag-value pairs it is the case that if one of the actions is executed, then eventually the other will be. As one can demonstrate on the level of the formal semantics, the left diagram does not fulfill this requirement because the Business may never Deliver Order. Also one can show that the right diagram does fulfill the requirement (intuitively, because the customer may reclaim the payment if the order is undelivered after the scheduled delivery date), assuming that the customer is able to prove having made the payment (indicated by the stereotype <<*provable*>>), e.g. by following a fair exchange protocol.

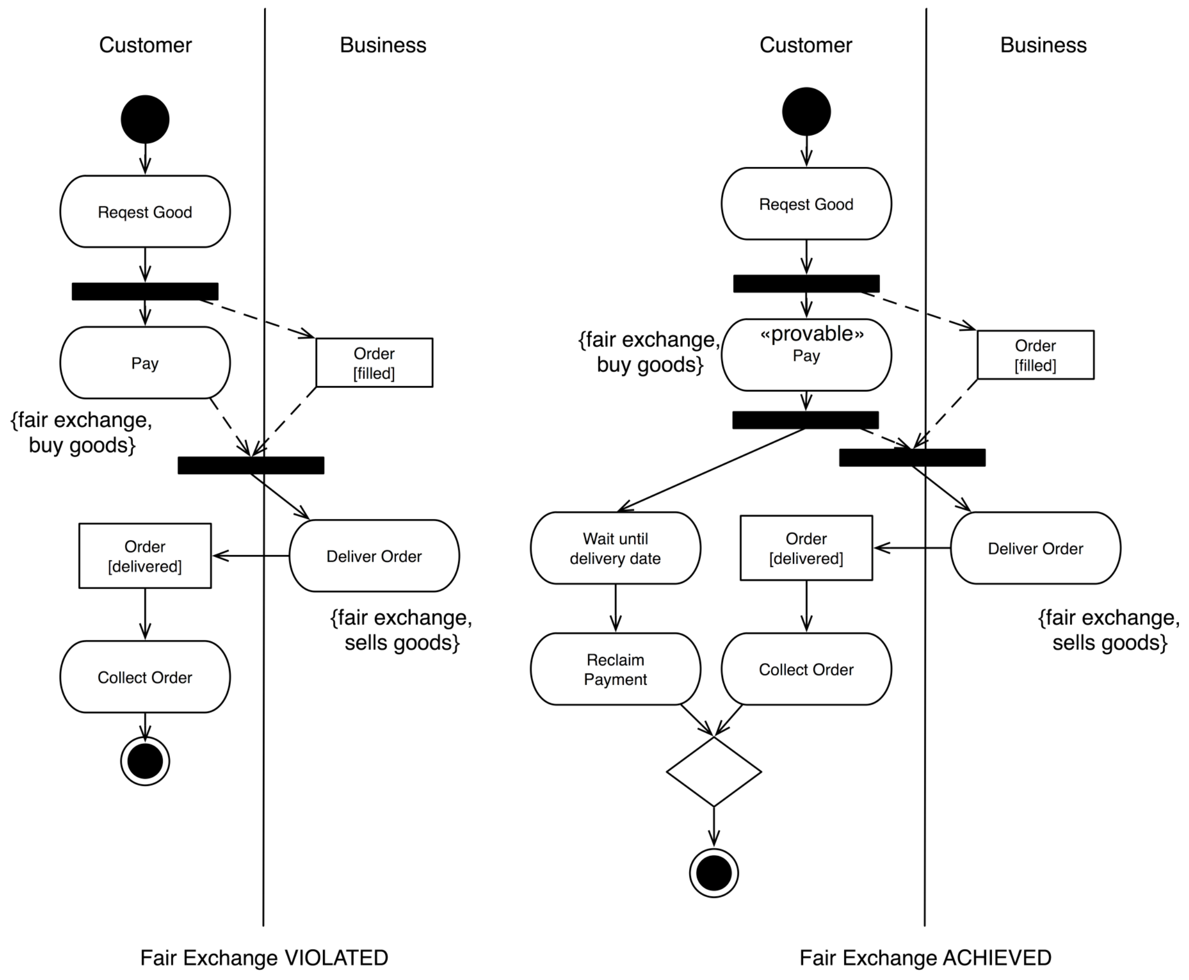


Figure 3: Activity diagram for "fair exchange"

Preservation of Sensitivity Levels with Class Diagrams

Class diagrams define the static structure of a system. As an example, Figure 4 gives a class-level description of a key generator (such as possibly used in the above business application example). The key generator offers the method newkey() which returns a Key for which it guarantees confidentiality and integrity.[1] On the other hand, it calls methods random() supposed to return a random number that is required to fulfill integrity and confidentiality (as specified in the model element stereotyped <<*outgoing actions*>> added in UMLsec). Here, this requirement is however not met by the random generator. As an example, consider the Homebanking-Computer-Interface (HBCI) specifications which in an early version (in the case of the RDH procedure) required the client system to perform key generation without specifying security requirements for the used random number generators. Omissions such as this one can be detected using our modelling approach.

[1] Here we use the convention that where the values are supposed to be boolean values, they need not be written (then presence of the label denotes the value true, and absence denotes false).

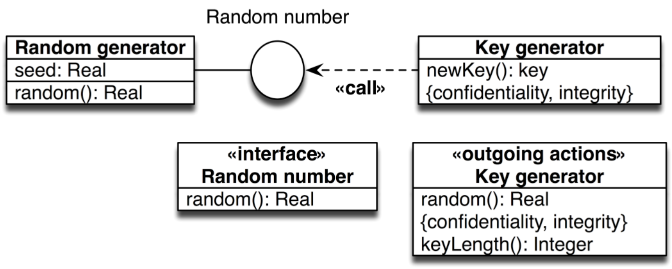


Figure 4: Class diagram for key generator

Here we assume that the attributes are fully encapsulated by the operations, i. e. an attribute of a class can be read and updated only by the class operations.

Security-Critical Message Exchange with Sequence Diagrams

In practice, the way security mechanisms (such as protocols) are employed in the system context offers more vulnerabilities than the mechanisms themselves. Also one sometimes has to adjust protocols to specific situations, e.g. for resource-bounded applications.

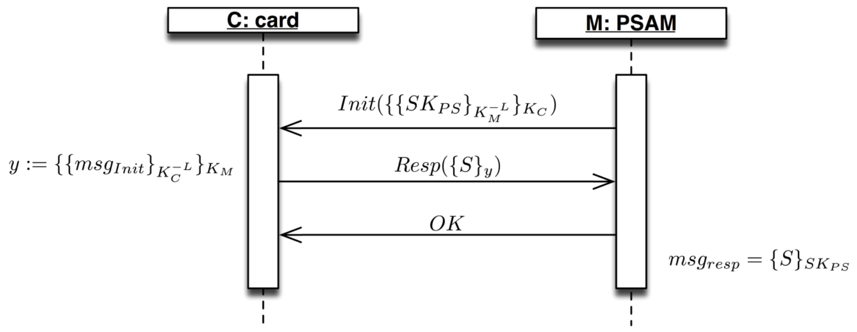

C: card
M: PSAM
$Init(\{\{SK_{PS}\}_{K_M^{-L}}\}_{K_C})$
$y := \{\{msg_{Init}\}_{K_C^{-L}}\}_{K_M}$
$Resp(\{S\}_y)$
OK
$msg_{resp} = \{S\}_{SK_{PS}}$

Figure 5: Sequence diagram for CEPS purchase transaction

In our UML-based approach, security protocols can be specified using message sequence charts. An example is given in Figure 5. Assumptions on the underlying physical layer (such as physical security of communication links) can be expressed in implementation diagrams (cf. below), and the behavior of the system context surrounding the protocol can be stated using state charts and reasoned about as indicated below. In Figure 5, two objects in the system (a card C and a purchase security application module (PSAM) M) exchange messages that involve cryptographic operations such as public-key encryption and signing. The encryption of the value *d* (or verification of signature) with the public key *K* is denoted by $\{d\}_K$, the decryption (or signing) with the private key K^{-1} by $\{d\}_{K-1}$ (for simplicity we assume use of RSA type encryption). Thus M starts by sending to C the message Init with argument a value SK_{PS} (the session key created by the PSAM) signed with M's private key and encrypted with C's public key. C decrypts the received message with its private key and verifies the signature with M's public key. C uses the resulting session key to encrypt a secret *S* which is then sent back as an argument of the message Resp M decrypts the received message using the session key and sends back the message OK.

Again one can make use of a formal semantics and automated tools to reason about such protocol descriptions.

Secure State Change with Statechart Diagrams

Statechart diagrams give the dynamic behaviour of an individual object: events may cause state in change or actions. We demonstrate how this kind of diagram can be used in the context of UMLsec with an example involving guards (such as used in Java security) in Figure 6. Statechart diagrams consist of states (written as boxes) and transitions between them. The initial state is marked with a transition leading out from a full circle. Transitions between states can be labelled with events, conditions (in square brackets) and actions (preceded by a backslash). An event can be a method of the specified object called by another object (e.g. the transition labelled checkGuard in Figure 6), and the interpretation is that the transition labelled with an event is fired if the event occurs while the object is in the state from which the transition goes out. If a transition is labelled by a condition (formulated in the UML-associated object constraint language (OCL) or otherwise) then it is only fired if the condition is true at the respective moment. If a transition is labelled with an action (which can be to call a method of another object or to assign a value to a local variable), then this action is executed whenever the transition is fired.

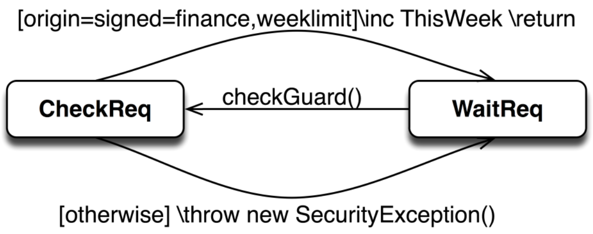

[origin=signed=finance,weeklimit]\inc ThisWeek \return
CheckReq
checkGuard()
WaitReq
[otherwise] \throw new SecurityException()

Figure 6: State chart for guard object

Suppose that a certain micropayment signature key may only be used by applets originating at and signed by the site Finance (e.g. to purchase stock rate information on behalf of the user), but this access should only be granted five times a week.

The Java 2 security architecture allows the use of guard objects for this purpose which regulate access to an object. In our example, the guard object can be specified as in Figure 6 (where ThisWeek counts the number of accesses in a given week and weeklimit is true if the limit has not been reached yet). One can then demonstrate using a formal semantics that certain access control requirements are enforced by the guards.

Assumptions on the Physical Layer with Deployment Diagrams

Deployment diagrams describe the underlying physical layer; we use them to ensure that security requirements on communication are met by the physical layer. For example, Figure 7 describes the physical situation underlying a system including a client system and a webserver communicating over the Internet. The two boxes labeled Client and Webserver denote nodes in the system (i. e. physical entities). The two rectangles labeled browser and access control represent system components residing on the respective nodes. The component browser has an interface offering the method get password. The component access control calls this method over the Internet via remote method invocation. The system specification requires the data exchanged in this invocation to be guaranteed confidentiality and integrity. However, these requirements are not met by the underlying communication link (as usual, this would be treated by using encryption).

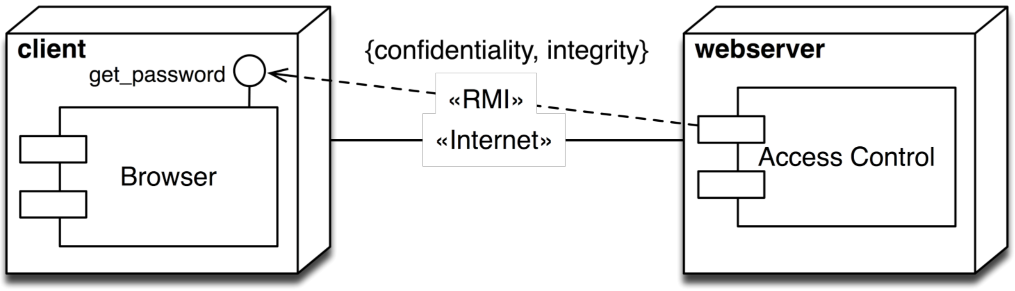


Figure 7: UMLsec deployment diagram

Code-level Assurance against High-level Security Requirements

Even if specifications exist for the implemented system, and even if these are formally analyzed, there is usually no guarantee that the implementation actually conforms to the specification. To deal with this problem, we can use the following approach: After specifying the system in the given notation (such as UMLsec) and verifying the model against the given security goals, we make sure that the implementation correctly implements the specification with techniques such as those explained below. Note that in addition it is often necessary to use dedicated tools to detect specialized weaknesses (such as buffer overflows), although this is not in scope of the current overview.

Run-time Security Monitoring: A simple and effective alternative is to insert security checks generated from the specification that remain in the code while in use, for example using the assertion statement that is part of the Java language (Bauer & Jürjens, 2008). These assertions then throw security exceptions when violated at run-time. In a similar way, this can also be done for C code. The SERENITY approach provides dynamic runtime verification mechanisms that can monitor various security properties dynamically based on event calculus (Maña et al., 2007). For example, they can monitor whether the assumptions made by a given security pattern is satisfied at the execution of the system. To achieve this, (Spanoudakis, Kloukinas & Androutsopoulos, 2007) proposes to use formal patterns that formalize frequently recurring system requirements as security monitoring patterns. Also, evolution tools record the operational data relevant for the Security and Dependability Patterns to obtain feedback that can help improving the patterns.

Model-based Test Generation: For performance-intensive applications, it may be preferable not to leave the assertions active in the code. This can be done by making sure by extensive testing that the assertions are always satisfied, for example by generating the test sequences automatically from the specifications. Since complete test coverage is often infeasible, an approach that automatically selects those test cases that are particularly sensitive to the specified security requirements is sketched in (Jürjens, 2009; Jürjens & Wimmel, 2002) (with respect to the formal semantics underlying UMLsec). Other work on testing crypto-protocols includes (Gürgens & Peralta, 2000).

Formally verifying cryptoprotocol implementations: For highly non-deterministic systems such as those using cryptography, testing can only provide assurance up to a certain degree. For higher levels of trustworthiness, it may therefore be desirable to establish that the code does enforce the security properties by a formal verification of the source code. There have recently been some approaches towards formally verifying implementations of crypto-protocols against high-level security requirements such as secrecy, for example (Bhargavan, Fournet, Gordon & Tse, 2006; Goubault-Larrecq & Parrennes, 2005; Jürjens, 2006; Jürjens & Yampolskiy, 2005). These works so far have aimed to verify implementations which were constructed with verification in mind (and in particular fulfill significant expectations on the way they are programmed) (Bhargavan et al., 2006; Goubault-Larrecq & Parrennes, 2005), or deal only with simplified versions of legacy implementations (Jürjens, 2006; Jürjens & Yampolskiy, 2005). In related work, (Pironti & Sisto, 2008) investigates under which conditions it is sound to abstract from marshalling and unmarshalling operations on transmitted messages when verifying protocol specifications.

Analyzing Security Configurations

There have also been some first steps towards linking model-based security engineering approaches with the automated analysis of security-critical configuration data. For example, a tool that automatically checks SAP R/3 user permissions for security policy rules formulated as UML specifications is presented in (Höhn & Jürjens, 2008). Because of its modular architecture and its standardized interfaces, the tool can be adapted to check security constraints in other kinds of application software, such as firewalls or other access control configurations.

Application Examples

An overview on industrial applications of model-based security engineering can be found in (Apvrille & Pourzandi, 2005). We list some examples below.

German Health Card architecture: Ongoing work for the German health telematics platform using a model-driven architectural framework and a security infrastructure based on Electronic Health Records and multifunctional Electronic Health Cards is presented in (Blobel & Pharow, 2007). A security analysis of the German Health Card Architecture using UMLsec is reported in (Jürjens & Rumm, 2008).

Electronic purses: UMLsec was applied to a security analysis of the Common Electronic Purse Specifications (CEPS), a candidate for a globally interoperable electronic purse standard supported by organizations representing 90 % of the world's electronic purse cards (including Visa International). Three significant security weaknesses were found in the purchase and load transaction protocols, improvements to the specifications were proposed, and it was shown that these are secure (Jürjens & Wimmel, 2001). There was also a security analysis of a prototypical Java Card implementation of CEPS. A method for the development of secure smartcard applications which includes UML models and is implemented in the KIV specification and verification system (Haneberg et al., 2002; Moebius et al., 2007) was applied to the specification of the Mondex electronic purse.

Intranet information systems: An application of UMLsec to information systems in an intranet at BMW is reported in (Best, Jürjens & Nuseibeh, 2007). There, the use of single-sign-on mechanisms was central, so the application of UMLsec was targeted to demonstrating that it was used correctly within the system context.

Biometric authentication: For a project with an industrial partner, UMLsec was chosen to support the development of a biometric authentication system at the specification level, where three significant security flaws were found (Jürjens, 2005b). It was also applied to the source-code level for a prototypical implementation constructed from the specification.

Web-based banking application: In a project with a German bank, model-based security engineering was applied to a web-based banking application to be used by customers to fill out and sign digital order forms (Jürjens, 2005a). The personal data in the forms must be kept confidential, and orders securely authenticated. The system uses a proprietary client authentication protocol layered over an SSL connection supposed to provide confidentiality and server authentication. Using the MBSE approach, the system architecture and the protocol were specified and verified with regard to the relevant security requirements.

Remaining Challenges in Model-Based Security

Given the current unsatisfactory state of computer security in practice, model-based security engineering seems a promising approach, since it enables developers who are not experts in security to make use of security engineering knowledge encapsulated in a widely used design notation. Since there are many highly subtle security requirements, which can hardly be verified with the "naked eye", even security experts may profit from this approach. Thus one can avoid mistakes that are difficult to find by testing alone, such as breaches of subtle security requirements, as well as the disadvantages of the "penetrate-and-patch" approach. Since preventing security flaws early in the system life cycle can significantly reduce costs, this gives a potential for developing securer systems in a cost-efficient way. Model-based security engineering has been successfully applied in various industrial projects. The approach has been generalized to other application domains such as real-time and dependability. Experiences show that the approach is adequate for use in practice, after relatively little training. As a consequence, model-based security engineering is now also considered an important emerging technology by industrial think-tanks. Due to space restriction, the current overview can only provide very limited detail or completeness. More comprehensive overviews on model-based security engineering and secure software engineering include (Jayaram & Mathur, 2005; Redwine, 2007). Some examples for open problems that remain:

Tracing security requirements: From a practical point of view, the construction of trustworthy security-critical systems would be significantly facilitated if one would have a practically feasible approach for tracing security requirements through the system lifecycle phases. A first step in that direction is presented in (Yu, Jürjens & Mylopoulos, 2008).

Preservation of security properties: Despite some early advances into this question (Jürjens, 2000; Jürjens, 2001a) there is so far relatively little known about the preservation of security properties when using design and analysis techniques such as the modular composition or decomposition, refinement or abstraction, or horizontal respectively vertical layering of system parts.

Security verification of legacy systems: A major open problem is to verify complex legacy implementations against high-level security properties in a practically feasible way. Again, some steps in that direction were reported above.

Security vs. other non-functional requirements / feature interaction: Another open problem is how to reconcile security requirements with other non-functional requirements, which may be orthogonal or even in conflict. First examples regarding performance properties can be found in (Maidl, Gilmore, Haenel & Kloul, 2005; Montangero, Buchholtz, Gilmore & Haenel, 2005; Woodside et al., 2009).

Model-based Identification of Vulnerabilities

Although software, requirements, and security engineering all offer methods for improved software development, today's software still contains vulnerabilities. These vulnerabilities, once exploited, can have various effects, ranging from subtle, maybe even more theoretical ones to practically devastating consequences. In a SOA, which implements business processes on top of web services, which again build upon software, it is evident that vulnerabilities jeopardize business processes. For that reason, this section shows how vulnerabilities can be identified. A brief introduction of the basic terms and concepts is followed by a discussion of the available approaches.

Vulnerabilities are flaws in information systems that can be abused to violate the security policy. The actual abuse is called an exploit. *Vulnerability analysis* aims to support avoiding, finding, fixing, and monitoring vulnerabilities (Bishop, 2005). These *vulnerability management* tasks typically require patterns, for example, to perform static or dynamic analysis of source code. Livshits (Livshits, 2006) presents an example of a practical method that uses such *vulnerability patterns*, specified as PQL queries (Martin, Livshits & Lam, 2005), to analyze Java programs for taint-style (i.e., use of unchecked input) vulnerabilities.

The *SOA* we refer to follows the OASIS SOA reference model and architecture (OASIS, 2006). The Business Process Execution Language (BPEL) is used to describe and execute the business processes. Processes in Business Process Modelling Notation (BPMN) can be translated to BPEL, e.g., through an Eclipse Plug-in developed by Google (http://code.google.com/p/bpmn2bpel/) Messages are exchanged through SOAP, and web service interfaces are defined in Web Service Description Language (WSDL). However, these standards and the WS Security standards family do not necessarily prevent software vulnerabilities from putting services at risk. We therefore refrain from a discussion of these standards and instead focus on software vulnerabilities and their model-based identification. Figure 8 shows our view on SOA layers, following, amongst others, Krafzig et al. (Krafzig, Banke & Slama, 2004) and IBM's reference architecture (Arsanjani, Zhang, Allam & Channabasavaiah, 2007). The bold arrows indicate the analysis focus: identifying vulnerabilities on the service level which result from software vulnerabilities.

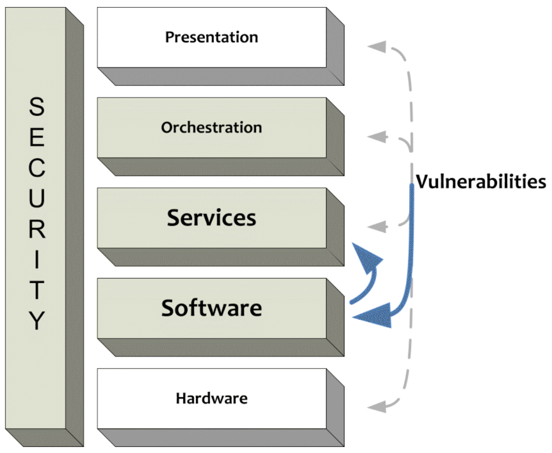


Figure 8: SOA Layers and Vulnerabilities

The presentation layer and the hardware layer will be using the same technology as before, so we do not analyze them any further. The software layer will increasingly be implemented using managed code, meaning a decrease of buffer overflows and similar vulnerabilities. Nevertheless, previous vulnerabilities might still be present in legacy software, so we share the view and classification of Yu et al. (Yu, Aravind & Supthaweesuk, 2006) for both the software and service layer. We differentiate between the service and the software layer by defining a service as a self-contained function which has a WSDL description and communicates through SOAP.

Business processes are executed as workflows, i.e., a set of activities performed in a certain order. The orchestration layer contains the workflow logic, typically implemented as an execution engine calling specific services in the order and with the parameters specified in a BPEL document. While before, the business process logic often was interwoven in PHP or ASP scripts of web applications, it is now separately defined in BPEL. Also, business process activities are no longer mere sections of code in a web application script, but specific services with a WSDL description and a SOAP interface. This creates new vulnerabilities in addition to the ones this section focuses on, the ones based in software applications. (Jensen, Gruschka, Herkenhöner & Luttenberger, 2007) lists some prominent examples with a description of counter measures. Regarding the services which implement the business process, two general options are available: top-down, generating services moving from the business process orchestration to the software layer, and bottom-up, wrapping existing software to be used in the service, and, consequently, in the orchestration. Given that initial SOA projects typically start by wrapping existing software, in the following we focus on the latter.

We employ the OASIS threat model (OASIS, 2006), comprising both internal and external attackers and the threats message alteration, message interception, man in the middle, spoofing, denial of service (DoS), replay, and false repudiation. Microsoft's STRIDE (Swiderski & Snyder, 2004) with its threats spoofing, tampering with data, repudiation, information disclosure, DoS, and elevation of privilege, is covered with that model because tampering with data, information disclosure, and elevation of privilege can be achieved through, e.g., message alteration and interception.

While in theory it might seem possible to come up with a complete list of vulnerabilities, in practice this is next to impossible even for simple components. In both cases, the precondition is that a certain task or attack goal is specified. Without a task that could be disturbed or an attack that is to be prevented, the most important search parameter for the vulnerability search would be missing (Anderson & Long, 2001). However, even if a method could determine all vulnerabilities regarding a certain task or attack, this list of tasks and attacks would have to be complete. Considering the fact that for the attacker it is enough to find a single attack the defenders did not think of (Anderson & Long, 2001), the initial idea of "simply" creating a complete list of vulnerabilities in an IT system turns out to be inapplicable. Zero-day vulnerabilities continue to be found (see (Turner, 2008) for current trends in zero-day vulnerabilities), which further indicates that complete vulnerability lists are a rather unrealistic assumption. The approach presented here is not based on the notion of such a complete list of vulnerabilities but on the notion of analyzing only known vulnerabilities the existence of which has been verified. Before discussing sources of such vulnerability information in the following, a brief description of the required information is given.

The main and mandatory detail about a vulnerability is where it is located, i.e., the name and version of the software it can be found in. Then, depending on the type of analysis, more or less information is required describing how the vulnerability can be exploited (e.g., remotely through a certain network protocol), what the direct effects are (e.g., elevation of privileges), and a severity rating (e.g., easily exploitable, severe effects on availability). The Common Vulnerability Scoring System (CVSS) (Mell, Scarfone & Romanosky, 2007) offers a widely accepted format for such descriptions (see Table 1).

CVSS Base Score: 0 (Low) to 10 (High) (Impact Subscore 0 to 10, Exploitability Subscore 0 to 10) **CVSS Temporal Score:** 0 (Low) to 10 (High) **CVSS v2 Vector:** (AV:N/AC:L/Au:N/C:C/I:C/A:C) **Published:** mm/dd/yyyy		
CVSS metric group	**CVSS metric**	**CVSS values**
Base Score Metrics		
Exploitability Metrics	Access Vector	local, adjacent network, network
	Access Complexity	low, medium, high
	Authentication	none, single, multiple
Impact Metrics	Confidentiality Impact	none, partial, complete
	Integrity Impact	none, partial, complete
	Availability Impact	none, partial, complete
Temporal Score Metrics	Exploitability	unproven, proof-of-concept, functional, high, not defined
	Remediation Level	official fix, temporary fix, workaround, unavailable, not defined
	Report Confidence	unconfirmed, uncorroborated, confirmed, not defined

Table 1: Common Vulnerability Scoring System

Sources of Vulnerability Information

In order to determine the effects an attacker could cause by exploiting vulnerabilities, a list of known vulnerabilities that are or could be present in the IT system at hand must be obtained. Penetration testing, source code analysis, and vulnerability databases allow compiling such a list of vulnerabilities and will now be discussed regarding their SOA applicability.

Penetration Testing

Also known as “red team testing”, penetration testing (Bishop, 2005; Thompson, 2005) is a hands-on approach to security analysis. Given a specific IT system, sometimes including configuration details, a team of security experts tries to attack the system and break the system’s security policy by, e.g., creating attack trees and then applying tools to check that the devised vulnerabilities actually are exploitable. Regarding attack trees, tools such as (Ou, Boyer & McQueen, 2006) can support the creation of graphs, however, care should be taken to not restrict the team’s focus to automatically generated graphs as attackers might come up with new ideas not covered by the graphs. Often, tools such as vulnerability scanners (see (Lyon, 2006) for a valuable list; Nessus and Microsoft’s Baseline Security Analyzer mark beginning and end of the top ten) and fuzzers are separately used to scan for vulnerabilities and to check how the system reacts to various inputs. The test result is a list of attacks the team was able to perform, along with detailed information on how the attacks were carried out. Besides configuration weaknesses and other information such as improper privilege settings, a very detailed description of vulnerabilities can be obtained.

With more and more exploit toolkits (e.g., Metasploit[i]) and other tools becoming available, penetration testing no longer is the tedious, purely manual task it used to be. Also, because the test is run on the actual IT system, the results are perfectly tailored to the specific setting. This is especially beneficial when a vulnerability has to be analyzed regarding its effect on the business process.

On the other hand, this advantage is a big drawback from a different point of view. Testing the real system does not only provide precise data on vulnerabilities and exploits, it can also damage the system. This is why penetration testing is not applicable in scenarios where, e.g., IT services may not be disturbed under any circumstances. In this case, a more theoretical approach can be used to analyze a model of the system.

Another drawback is that employing a team of security experts is expensive, sometimes prohibitively. This fact links to the additional drawback that because it is so expensive, most companies cannot afford continuous penetration tests. Therefore, it is not suitable to maintain an up-to-date list of vulnerabilities. Penetration testing still is very valuable in terms of the highly detailed and relevant vulnerability data it creates.

Source Code Analysis

Besides manual code inspection, elaborate algorithms and tools are able to find vulnerabilities in source code. Tool support started with rather simple lexical scanners (Chess & McGraw, 2004), which, for example, point out the use of *printf* function calls in C so that a human analyst can (and has to) decide whether there is a buffer overflow vulnerability or not. Far more sophisticated approaches use model-checking to show that, e.g., there is no path in the code an input string could reach a certain part of the code without being sanitized. (Livshits, 2006) presents such an approach for Java web applications. It offers a sound solution which includes pointer and reflection analysis, meaning that a call graph is constructed which allows tracing input data along functions calls including, to some extent, code or libraries which are not statically linked.

Source code analysis does not touch the actual system and thus, unlike penetration testing, will not interrupt any services. For the same reason it produces results of a slightly different quality, because the code will never run alone but always in combination with other software. To illustrate what this means, consider a web service with an input vulnerability which will crash the service as soon as an overly long input is received. If XML schema validation is in place (see (Jensen et al., 2007) for an overview of attacks that schema validation can defend against), dropping the SOAP message with the dangerous XML input, source code analysis of the web service would report a vulnerability, while penetration testing would not show it (except if XML schema validation had been disabled through an earlier attack during testing).

In general and from a practical point of view, source code analysis is more likely to yield false positives than penetration testing. This must be kept in mind when assessing the vulnerability effects as described below in the evaluation section. The forensic approach based on logging will filter the more theoretical vulnerabilities source code analysis reports. Achieving the same filtering with the model-based approach is possible yet imposes additional complexity on the model as more components must be incorporated.

Vulnerability Databases

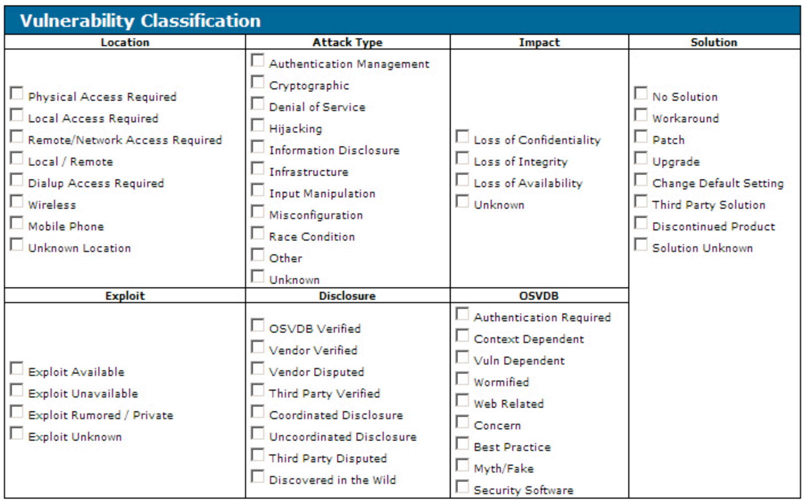
Querying vulnerability databases to benefit from the security community's vulnerability search efforts is an approach that should always be applied. Even if a company has enough resources to perform its own penetration testing and source code analysis, it makes little sense to ignore the vulnerabilities others have found and published. Because the results of a company-internal vulnerability search can be entered into a database as well, the vulnerability identification suggested here does not specifically build upon penetration testing or source code analysis, but relies on vulnerability databases as the more general solution. So, besides company-internal databases, which vulnerability databases are available? We briefly describe some sources of vulnerability information which we did not consider, and then describe the more suitable ones.

While most databases list vulnerabilities independent of the software vendor, *Microsoft*'s vulnerability database[ii] only lists vulnerabilities in Microsoft software. However, the descriptions contain all the standard entries discussed below. *SecWatch*[iii] can be considered a meta search engine, it does not provide its own description scheme. In a similar manner, the *SANS Newsletter*[iv] offers a brief vulnerability overview and refers to other databases for the details. The *Open Web Application Security Project (OWASP)*[v] does not list vulnerability instances but vulnerability types. It might serve as a guide for a penetration test or source code analysis, but it does not contain vulnerabilities in actual products, services, or web applications. *Milw0rm*[vi] does not offer its vulnerability descriptions in a standardized format, therefore it was excluded from the comparison. The same holds true for the SecuriTeam database[vii], which in addition draws many descriptions from the other databases.

The following free online databases will be compared below (in alphabetical order): *French Security Incident Response Team (FRSIRT)*[viii], *Internet Security Systems (ISS) X-Force*[ix], *National Vulnerability Database (NVD)*[x] with *Common Vulnerabilities and Exposures (CVE)*[xi] entries, *Open Source Vulnerability Database (OSVDB)*[xii], *SecurityFocus*[xiii] (including *Buqtraq*), and *SecurityTracker*[xiv].

Comparing the above-mentioned databases, it shows that the majority has a big overlap between their description schemes. All databases offer their own and unique name or id for each vulnerability, a disclosure date, and a textual vulnerability description. In addition, they all list the vulnerable software including version numbers, credits or pointers to the information source, references to related reports and descriptions in other databases, references to the affected software's vendor, and protection hints or links to such.

CVE names are included in all description schemes, thus CVE and the corresponding CVSS data can easily be obtained. FRSIRT and SecurityFocus explicitly describe the access vector in their descriptions ("remote/local" with "yes/no" values each), perhaps in order to not depend on the completeness of CVSS data. OSVDB, Secunia, and SecurityFocus each use their own vulnerability classification (see Table 2 for an example). These specific classifications can in some cases aid in determining the effect a vulnerability's exploit could have, however, they require a customized analysis and should be considered volatile as they do not follow a widely-accepted standard such as CVSS.



Vulnerability Classification

Location	Attack Type	Impact	Solution
☐ Physical Access Required ☐ Local Access Required ☐ Remote/Network Access Required ☐ Local / Remote ☐ Dialup Access Required ☐ Wireless ☐ Mobile Phone ☐ Unknown Location	☐ Authentication Management ☐ Cryptographic ☐ Denial of Service ☐ Hijacking ☐ Information Disclosure ☐ Infrastructure ☐ Input Manipulation ☐ Misconfiguration ☐ Race Condition ☐ Other ☐ Unknown	☐ Loss of Confidentiality ☐ Loss of Integrity ☐ Loss of Availability ☐ Unknown	☐ No Solution ☐ Workaround ☐ Patch ☐ Upgrade ☐ Change Default Setting ☐ Third Party Solution ☐ Discontinued Product ☐ Solution Unknown
Exploit	**Disclosure**	**OSVDB**	
☐ Exploit Available ☐ Exploit Unavailable ☐ Exploit Rumored / Private ☐ Exploit Unknown	☐ OSVDB Verified ☐ Vendor Verified ☐ Vendor Disputed ☐ Third Party Verified ☐ Coordinated Disclosure ☐ Uncoordinated Disclosure ☐ Third Party Disputed ☐ Discovered in the Wild	☐ Authentication Required ☐ Context Dependent ☐ Vuln Dependent ☐ Wormified ☐ Web Related ☐ Concern ☐ Best Practice ☐ Myth/Fake ☐ Security Software	

Table 2: OSVDB's Vulnerability Classification

In terms of vulnerability effect determination, ISS X-Force's database is particularly interesting as it contains adjusted CVSS data (CVSS temporal values filled in by the ISS team) and a "business impact" field, containing a textual description of the business impact. To give an example, CVE-2008-3466 business impact according to ISS reads: "Compromise of networks and machines using affected versions of Microsoft Host Integration Server may lead to exposure of confidential information, loss of productivity, and further network compromise. An attacker does not need to entice any kind of user interaction to trigger this vulnerability. Successful exploitation would grant an attacker the privileges of the SNA RPC service." Even though the description of effects such as loss of confidentiality also are part of CVSS data, the brief information in this field allows a quick assessment of the vulnerability and represents a promising, business-oriented extension of the description scheme.

Identifying Vulnerabilities in Services

Vulnerable software can lead to vulnerable services, which again can lead to vulnerable business processes as, in a SOA, these are implemented as composition of services. Having obtained a list of vulnerabilities as described above, this information must now be mapped to services and, ultimately, the business process. In the following, the focus lies on mapping between software and services, as the mapping between services and business processes is directly contained in BPMN or BPEL documents.

The vulnerability databases mentioned above contain a large number of vulnerability descriptions that refer to web related vulnerabilities. Nevertheless, there are very few entries on vulnerabilities in actual web services. While there will be more such entries as more web services become available, the software currently deployed will still be used and wrapped in web services. Therefore, given the description of a software vulnerability, the question is which services build upon that software and, thus, could be affected by an exploit of the vulnerability. To answer that question, we extract the required information from vulnerability descriptions and include it in the UMLsec model of the system under analysis. This allows a model-based vulnerability identification, as Figure 9 shows.

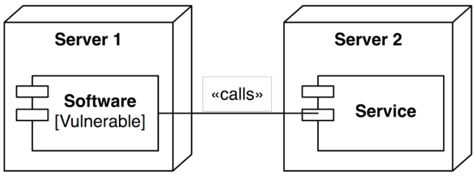
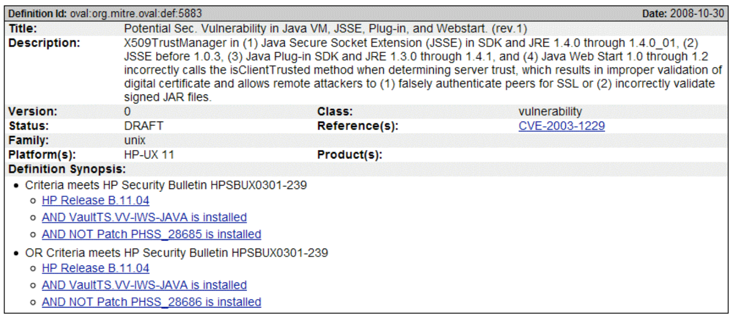


Figure 9: UML deployment diagram with software-service mapping

Vulnerability descriptions always include the target software, i.e. the software the vulnerability was found in. To avoid false positives during the analysis, a preliminary check should be run regarding the current IT environment: If the software has been patched already or a newer version has been installed which does not contain the vulnerability, the vulnerability should be excluded from the list before being mapped to services. The Open Vulnerability and Assessment Language (OVAL) (MITRE, 2007) can be used to obtain the information necessary to run such a check. Table 3 shows an exemplary OVAL definition.



Definition Id: oval:org.mitre.oval:def:5883			**Date:** 2008-10-30
Title:	Potential Sec. Vulnerability in Java VM, JSSE, Plug-in, and Webstart. (rev.1)		
Description:	X509TrustManager in (1) Java Secure Socket Extension (JSSE) in SDK and JRE 1.4.0 through 1.4.0_01, (2) JSSE before 1.0.3, (3) Java Plug-in SDK and JRE 1.3.0 through 1.4.1, and (4) Java Web Start 1.0 through 1.2 incorrectly calls the isClientTrusted method when determining server trust, which results in improper validation of digital certificate and allows remote attackers to (1) falsely authenticate peers for SSL or (2) incorrectly validate signed JAR files.		
Version:	0	**Class:**	vulnerability
Status:	DRAFT	**Reference(s):**	CVE-2003-1229
Family:	unix		
Platform(s):	HP-UX 11	**Product(s):**	
Definition Synopsis:			

- Criteria meets HP Security Bulletin HPSBUX0301-239
  - HP Release B.11.04
  - AND VaultTS.VV-IWS-JAVA is installed
  - AND NOT Patch PHSS_28685 is installed
- OR Criteria meets HP Security Bulletin HPSBUX0301-239
  - HP Release B.11.04
  - AND VaultTS.VV-IWS-JAVA is installed
  - AND NOT Patch PHSS_28686 is installed

Table 3: Exemplary OVAL definition

Once the list of actual vulnerabilities in a specific IT system has been created, and all vulnerabilities that have been patched or removed in other ways have been excluded, the services that run on top of the affected software must be identified. This identification step can be done before, at, or after runtime, and each of these options will be discussed below.

Identification Before Runtime

During service design time, it is apparent which other services and software a certain service involves. A simple approach is entering these mappings into a database. Then, each time a vulnerable service or software is to be checked for connected services, this database can be queried. However, a clear drawback of this simple approach is that it highly depends on the designer to enter each and every mapping. This can be remedied by integrating the mapping into the implementation phase. When implementing a service, the supporting services and underlying applications must be specified. In integrated development environments (IDE) such as JDeveloper [xv], every time such a connection is created the target can be selected from a constantly updated list of available services. Also, the programmer can select databases and other software the service will connect to. Keeping track of these assignments allows to map between vulnerable software (and services) and the services which involve those vulnerable components.

Regardless of a purely manual or partly automated approach of mapping before runtime, both the mapping between software and services and the mapping between services is covered. The manual approach will impose little additional effort during SOA creation; however, it is too cumbersome to be applied when an existing SOA is frequently changed. IDE support for service mapping is already implemented today (e.g., in JDeveloper), support for software to service mapping so far is rudimentary but, in IDEs such as Eclipse[xvi], can be added through plugins.

Having obtained the mapping between software and services, these links can be included in the UMLsec model. Then, each time a new vulnerability is announced, the model can be automatically analysed, resulting in an identification of the affected services. Because in a SOA, service compositions are the BPMN or BPEL equivalent of a business process, this model-based approach provides pre-runtime vulnerability identification for SOA-based business processes.

Identification At Runtime

Web Services communicate with each other through SOAP messages. A network monitor observing these messages can gather information on the hosts and ports the services are running on. The problem here is that while sometimes this information is sufficient to automatically construct a mapping table (compare Figure 10a), in other cases the network traffic only allows ambiguous conclusions (cf. Figure 10b).

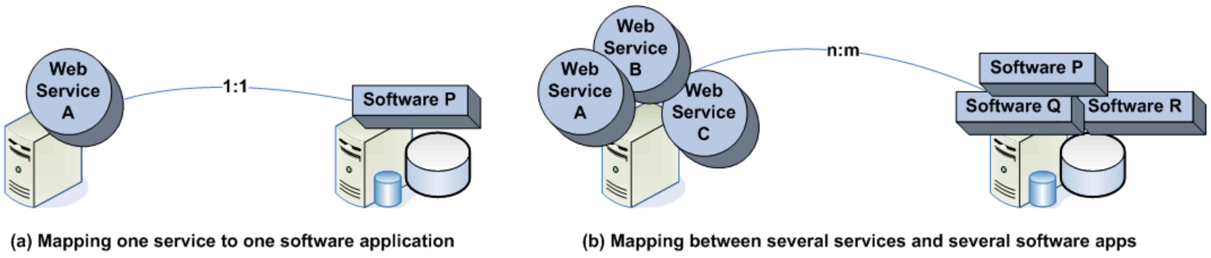

Web Service A
1:1
Software P
(a) Mapping one service to one software application
Web Service B
Web Service A
Web Service C
n:m
Software P
Software Q
Software R
(b) Mapping between several services and several software apps

Figure 10: Identification of the software that a service uses

Imagine a network in which each host either supports exactly one service or software. Seeing a communication between a service host and a software host, the monitor can deduce that the service depends on the software (cf. Figure 10a). Now imagine a network with only two hosts, one running all the services, and one running all the software. Unless analyzing the protocols in use allows a precise service and software identification, the monitor can only guess which service uses which software (cf. Figure 10b). So if the monitor sees traffic from different services going to a host with multiple software applications, it cannot always automatically map between services and software applications. For example, automated mapping is possible when a service queries a MySQL database with a constant, unique username. It is impossible when several services use the same username for their queries. Note that SQL injection vulnerabilities can easily be a source of false positives when analyzing a SOA: If the service correctly filters its input before sending it to the database, the vulnerability is not exploitable through the service. However, assuming that SQL injections cannot be executed just because software is wrapped into a service is a fallacy.

Ambiguous communication only exists when software is involved. Services communicating with each other can be identified through their unique name in SOAP messages. This means that the runtime approach can automatically identify service-to-service mappings. For software to service mappings, sometimes a human will have to resolve ambiguities. In restricted cases an automated identification will be possible, yet, in the general case, mapping at runtime might require manual intervention.

Again, as in the pre-runtime case, once the mapping of software to services is captured, it serves as input to the UMLsec model and thusly enables model-based identification of vulnerabilities in a SOA-based business process. The drawback is that here, runtime behaviour must be observed before the analysis can begin, which opens a window of opportunity for attackers. Nevertheless, the runtime approach is valuable when the previous approach is not applicable, and it can also serve to confirm the findings of the pre-runtime mapping.

Identification After Runtime

Log files offer an additional opportunity of identifying the links between services and the software they use. Depending on the network setup and the level of detail found in the logs, correlating log entries to map from services to software can be straightforward or prove rather difficult. Therefore, the applicability of this approach is high when the logs already offer detailed information, or the logging mechanisms can easily be extended to include details on services and software being invoked. In a highly complex setting, where customized logging cannot be implemented, and the logfiles at hand offer too low a level of detail, mapping after runtime requires manual involvement. Such manual resolution of mapping ambiguities might seem acceptable in a scenario with static links. However, one of the main notions behind web services is flexibility, so that the assumption of static links will rarely hold in practice.

The model-based identification of vulnerabilities after runtime can be performed through an audit of log files. To this end, one must first ensure that system records are written and stored in a secure way. Here, we assume that the storage provides confidentiality and tamper-evidence. Confidentiality means that entries stored in the log file cannot be read by unauthorized subjects and is necessary to avoid the replication and the replay of entries. Tamper-evidence means that if logfiles are manipulated, then these manipulations, e.g. deletion of entries or entry modification, are *evident* to a verifier. Technically, tamper-evidence is broken down into three subordinated properties: all the entries must be authentic, i.e. they are stored, as they have been transmitted from the logging device to the logging system. The entries must be encrypted in a forward secure way, i.e. knowledge of the key to decrypt one of the entries may not put an attacker in a position where he can decrypt all – in particular the previous – the entries. And the entries must be complete, i.e. deletion of one or more entries must be detected by the system.

These properties are achieved by secure logging protocols. Current state of the art can be distinguished into those protocols adding confidentiality and tamper-evidence to the traditional syslog and those designed for providing these functionalities from the outset based on the Schneier-Kelsey secure logging protocol scheme (Schneier & Kelsey, 1999). Thorough threat analysis has demonstrated that these protocols fail to provide the necessary guarantees, so that attackers can still manipulate with log data (Accorsi, 2008).

To our knowledge, the only secure logging protocol providing for all the necessary security properties is that proposed in (Accorsi, 2006). In this protocol suite, confidentiality is achieved using an evolving encryption scheme (Franklin, 2006): the current encryption key is newly calculated for every entry using a one-way function. An attacker that can get hold of the encryption key for one entry *n* is not able to calculate the key for the previous entries *n-1*, *n-2,…*. Clearly, it is possible to calculate the keys for the log entries *n+1, n+2,…*. This property is not overly important though, as a logging-system that has been corrupted cannot write secure and reliable log messages anyway. With regard to tamper evidence, entries are digitally signed and, therefore, cannot be modified without knowing the private key used to generate these digital signatures. In addition to this, a hash chain that connects all the entries with each other guarantees the completeness of the entries. (See (Lamport & Schneider, 1984) for details on hash-chains.) This hash chain is calculated for an entry by applying some cryptographic hash functions to the previous entry. Only if the chain is correct from the first to the last entry one can be sure that the log file has not been tampered: a broken chain indicates that tampering has taken place.

One aspect that if often overlooked when applying hash-chains in this setting is that one must check whether the number of entries is correct. The proposed scheme is not able to detect truncation attacks where an attacker removes entries from the tail of the hash-chain. This can easily be calculated, by comparing the current cryptographic key (which is calculated by applying *n*-times the evolution function, with *n* being the total number of entries in the file) with the actual length of the log file.

Such secure logging mechanisms provide a sound basis for the evaluation of vulnerability effects. The actual evaluation takes place by means of an audit, whose goal is to correlate suspicious events recorded in the log file and check whether the emerging patterns matches to one (or more) in the vulnerability database. Such an audit can efficiently be implemented using, e.g., a pruning algorithm based on the structure of the log files (Accorsi & Stocker, 2008). Together with a normalization step, which merely correlates events, the algorithm runs in linear time and, hence, can cope with log files of industrial size. An alternative approach is proposed in (Accorsi, 2008).

In order to carry out such an automated audit, one needs a formalization of the effects of the vulnerabilities. In (Accorsi & Stocker, 2008), a Domain Specific Language based on XML is employed when expressing the effects. Specifically, this language was designed to capture the different preconditions for and postconditions arising from successful exploits. These conditions can be derived from vulnerability descriptions, so that an automated audit becomes possible. For example, CVSS entries describe the access vector to a vulnerability, which represents a precondition, and the confidentiality/integrity/availability impact, which represent postconditions. Given the log files and the pre- and postconditions, pattern matching on the log data can pinpoint vulnerable services and indicate the effects an exploit of a certain vulnerability might induce on the services and their composition. Because this composition represents the implementation of a business process, the analysis result is a list of those parts of the business process that are put at risk by vulnerabilities.

Obviously, this “after the fact” approach opens the biggest window of opportunity for attackers. However, log-based audits are necessary to verify that the system works as expected. And even if this approach does not prevent a specific vulnerability from being exploited, the identification after runtime points that vulnerability out so that it can be fixed. This clearly provides better security than leaving the services and their composition vulnerable.

Discussion

The current state of vulnerability analysis tools clearly shows that identification of existing vulnerabilities is crucial for every systematic approach. Although we have presented an approach that can cope with the impact of distinct vulnerabilities, still no systematic way of identifying unknown vulnerabilities exists. Furthermore, it is important to realize that WS-Security and similar standards will not reduce the number of vulnerabilities in a complex business infrastructure. Because they cause additional complexity, they much rather introduce new vulnerabilities into the computer systems.

The exact determination of the effects a vulnerability exploit can have still is an open issue. Certainly, a viable solution is to use forensic information as a reliable source of information. But it is necessary to consider the attackers’ intelligence and knowledge, which might cause new, unforeseen effects. Attackers can find completely new attack paths in the system, which the system providers did not think of, let alone protect. To this end, it is necessary to closely investigate the different effects of vulnerabilities and their categorization. Such a categorization will allow for systematic verification of the absence of certain attack paths.

Even with such verified absence of certain attack paths, what is still missing is a mechanism to protect systems from zero day exploits. The simple reason is the fact that it is impossible to verify a given computer system regarding *unknown* properties. If the zero day exploit falls into one of the well-known categories and it is proven that vulnerabilities within this category cannot cause any damage, then the new zero day exploit will not cause any damage. Newly discovered vulnerabilities that fall into a different category still impose a threat.

Another tricky aspect of the methodology proposed in this chapter is the area of conflict between generic evaluation of processes (or process patterns) and the specific effects that are observed in a specific instance of a business process. In many cases, it is hardly possible to predict the effects of exploited vulnerabilities without taking into account the specific business goals. Further analysis is required regarding the degree to which the effects of exploiting a given vulnerability depend on the business process under consideration.

References

Accorsi, R. (2006). On the relationship of privacy and secure remote logging in dynamic systems. In *Proceedings of the international information security conference* . (pp. 329-39). Boston: Springer-Verlag.

Accorsi, R. (2008). *Automated counterexample-driven audits of authentic system records.* Thesis,

Accorsi, R., & Stocker, T. (2008). Automated privacy audits based on pruning of log data. In *Proceedings of the international workshop on security and privacy in enterprise computing.* IEEE Computer Society.

Agreiter, B., Alam, M., Hafner, M., Seifert, J. P., & Zhang, X. (2007). Model driven configuration of secure operating systems for mobile applications in health care. In J. Sztipanovits, R. Breu, E. Ammenwerth, R. Bajcsy, J. Mitchell, & A. Pretschner (Eds.), *Workshop on model-based trustworthy health information systems.*

Alam, M., Breu, R., & Hafner, M. (2007). Model-Driven security engineering for trust management in SECTET. *Journal of Software*, *2*(1), 47-59.

Anderson, R., & Long, C. (2001). *Security engineering: A guide to building dependable distributed systems.* Wiley & Sons.

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

Arenas, A., Aziz, B., Bicarregui, J., Matthews, B., & Yang, E. Y. (2008). Modelling security properties in a grid-based operating system with anti-goals. In *ARES '08: Proceedings of the 2008 third international conference on availability, reliability and security.* IEEE Computer Society.

Arsanjani, A., Zhang, L. J., Allam, A., & Channabasavaiah, K. (2007, March 28). *Design a SOA solution using a reference architecture*. IBM. Retrieved April 20, 2009, from http://www.ibm.com/developerworks/library/ar-archtemp/

Baldwin, A., Beres, Y., Shiu, S., & Kearney, P. (2006). A model-based approach to trust, security and assurance. *BT Technology Journal*, *24*(4), 53-68.

Basin, D., Doser, J., & Lodderstedt, T. (2006). Model driven security: From UML models to access control infrastructures. *ACM Trans. Softw. Eng. Methodol.*, *15*(1), 39-91.

Bauer, A., & Jürjens, J. (2008). Security protocols, properties, and their monitoring. In *SESS '08: Proceedings of the fourth international workshop on software engineering for secure systems.* ACM.

Best, B., Jürjens, J., & Nuseibeh, B. (2007). Model-Based security engineering of distributed information systems using UMLsec. In *ICSE '07: Proceedings of the 29th international conference on software engineering.* IEEE Computer Society.

Bhargavan, K., Fournet, C., Gordon, A. D., & Tse, S. (2006). Verified interoperable implementations of security protocols. In *Computer security foundations workshop, 2006 19th IEEE.*

Bishop, M. (2005). *Introduction to computer security.* Addison-Wesley.

Blobel, B., & Pharow, P. (2007). A model driven approach for the german health telematics architectural framework and security infrastructure. *International Journal of Medical Informatics*, *76*(2-3), 169-175.

Blobel, B., Nordberg, R., Davis, J. M., & Pharow, P. (2006). Modelling privilege management and access control. *International Journal of Medical Informatics*, *75*(8), 597-623.

Boehm, B. W. (1981). *Software engineering economics.* Prentice-Hall Englewood Cliffs, NJ.

Chess, B., & McGraw, G. (2004). Static analysis for security. *IEEE Security and Privacy*, *2*(6), 76-79.

Crook, R., Ince, D., Lin, L., & Nuseibeh, B. (2002). Security requirements engineering: When anti-requirements hit the fan. In *Requirements engineering, 2002 proceedings IEEE joint international conference on.*
Deubler, M., Grünbauer, J., Jürjens, J., & Wimmel, G. (2004). Sound development of secure service-based systems. In *Proceedings of the 2nd international conference on service oriented computing.*
Devanbu, P. T., & Stubblebine, S. (2000). Software engineering for security: A roadmap. In *ICSE '00: Proceedings of the conference on the future of software engineering.* ACM.
Dimitrakos, T., Ritchie, B., Raptis, D., Aagedal, J., Braber, F. D., Stølen, K., et al. (2002). Integrating model-based security risk management into ebusiness systems development: The CORAS approach. In *I3E '02: Proceedings of the IFIP conference on towards the knowledge society.* Kluwer, B.V.
Eckert, C., & Marek, D. (1997). Developing secure applications: A systematic approach. In *SEC'97: Proceedings of the IFIP TC11 13 international conference on information security (SEC '97) on information security in research and business.* Chapman Hall, Ltd.
Egea, M., Basin, D., Clavel, M., & Doser, J. (2007). A metamodel-based approach for analyzing security-design models. In *Lecture notes in computer science.* (pp. 420-35). Springer Berlin / Heidelberg.
Fernandez, E. B., & Hawkins, J. C. (1997). Determining role rights from use cases. In *RBAC '97: Proceedings of the second ACM workshop on role-based access control.* ACM.
Fernandez, E. B., Larrondo-Petrie, M. M., Sorgente, T., & VanHilst, M. (2006). A methodology to develop secure systems using patterns. *Integrating Security and Software Engineering: Advances and Future Vision*, 107-126.
Flechais, I., Mascolo, C., & Sasse, M. A. (2007). Integrating security and usability into the requirements and design process. *International Journal of Electronic Security and Digital Forensics*, *1*(1), 12-26.
Fox, J., Mouratidis, H., & Jürjens, J. (2006). Towards a comprehensive framework for secure systems development. In *Lecture notes in computer science.* (pp. 48-62). Springer Berlin / Heidelberg.
Franklin, M. (2006). A survey of keyx evolving cryptosystems. *International Journal of Security and Networks*, *1*(1/2), 46-53.
Giorgini, P., Massacci, F., Mylopoulos, J., & Zannone, N. (2005). Modeling security requirements through ownership, permission and delegation. In *Requirements engineering, 2005 proceedings 13th IEEE international conference on.*
Gollmann, D. (2000). On the verification of cryptographic protocols: A tale of two committees. *Electronic Notes in Theoretical Computer Science*, *32*, 42-58.
Goubault-Larrecq, J., & Parrennes, F. (2005). Cryptographic protocol analysis on real C code. In *Verification, model checking, and abstract interpretation.* Springer Berlin / Heidelberg.
Gürgens, S., & Peralta, R. (2000). Validation of cryptographic protocols by efficient automated testing. In *Proceedings of the thirteenth international florida artificial intelligence research society conference.* AAAI Press.
Hafner, M., Memon, M., & Alam, M. (2008). Modeling and enforcing advanced access control policies in healthcare systems with sectet. In *Models in software engineering .*
Haley, C. B., Laney, R., Moffett, J. D., & Nuseibeh, B. (2008). Security requirements engineering: A framework for representation and analysis. *Transactions on Software Engineering*, *34*(1), 133-153.
Haneberg, D., Reif, W., & Stenzel, K. (2002). A method for secure smartcard applications. In *AMAST '02: Proceedings of the 9th international conference on algebraic methodology and software technology.* London, UK: Springer-Verlag.
Houmb, S. H., Georg, G., France, R., Bieman, J., & Jürjens, J. (2005). Cost-Benefit trade-off analysis using BBN for aspect-oriented risk-driven development. In *Engineering of complex computer systems, 2005 ICECCS 2005 proceedings 10th IEEE international conference on.*
Höhn, S., & Jürjens, J. (2008). Rubacon: Automated support for model-based compliance engineering. In *ICSE '08: Proceedings of the 30th international conference on software engineering.* ACM.
Hultin, F., & Heldal, R. (2003). Bridging model-based and language-based security. In *Lecture notes in computer science.* (pp. 235-52). Springer Berlin / Heidelberg.
Jayaram, K. R., & Mathur, A. P. (2005). Software engineering for secure software - state of the art: A survey. *CERIAS and SERC SERC-TR-279, September 19Th.*

Jensen, M., Gruschka, N., Herkenhöner, R., & Luttenberger, N. (2007). SOA web services: New technologies - new standards - new attacks. In *Proceedings of the 5th IEEE european conference on web services (ECOWS).*
Jürjens, J. (2000). Secure information flow for concurrent processes. In *Lecture notes in computer science.* (pp. 395-409). Springer Berlin / Heidelberg.
Jürjens, J. (2001a). Secrecy-Preserving refinement. In *Lecture notes in computer science.* (pp. 135-52). Springer Berlin / Heidelberg.
Jürjens, J. (2001b). Towards development of secure systems using UMLsec. In *Lecture notes in computer science.* (pp. 187-200). Springer Berlin / Heidelberg.
Jürjens, J. (2002). UMLsec: Extending UML for secure systems development. In *UML '02: Proceedings of the 5th international conference on the unified modeling language.* London, UK: Springer-Verlag.
Jürjens, J. (2005a). *Secure systems development with UML.* Springer.
Jürjens, J. (2005b). Sound methods and effective tools for model-based security engineering with UML. In *Software engineering, 2005 ICSE 2005 proceedings 27th international conference on.*
Jürjens, J. (2006). Security analysis of crypto-based java programs using automated theorem provers. In *ASE '06: Proceedings of the 21st IEEE/ACM international conference on automated software engineering.* IEEE Computer Society.
Jürjens, J. (2009). A domain-specific language for cryptographic protocols based on streams. *Journal of Logic and Algebraic Programming*, 54-73.
Jürjens, J., & Rumm, R. (2008). Model-Based security analysis of the german health card architecture. *Methods of Information in Medicine*, *47*(5), 409-416.
Jürjens, J., & Shabalin, P. (2007). Tools for secure systems development with UML. *Int. J. Softw. Tools Technol. Transf.*, *9*(5), 527-544.
Jürjens, J., & Wimmel, G. (2001). Security modelling for electronic commerce: The common electronic purse specifications. In *I3E '01: Proceedings of the IFIP conference on towards the e-society.* Kluwer, B.V.
Jürjens, J., & Wimmel, G. (2002). Specification-Based test generation for security-critical systems using mutations. In *Lecture notes in computer science.* (pp. 471-82). Springer Berlin / Heidelberg.
Jürjens, J., & Yampolskiy, M. (2005). Code security analysis with assertions. In *ASE '05: Proceedings of the 20th IEEE/ACM international conference on automated software engineering.* ACM.
Kearney, P., & Brügger, L. (2007). A risk-driven security analysis method and modelling language. *BT Technology Journal*, *25*(1), 141-153.
Koch, M., & Parisi-Presicce, F. (2006). UML specification of access control policies and their formal verification. *Software and Systems Modeling*, *5*(4), 429-447.
Krafzig, D., Banke, K., & Slama, D. (2004). *Enterprise SOA: Service-Oriented architecture best practices.* Prentice Hall PTR.
Lamport, L., & Schneider, F. (1984). The 'hoare logic' of CSP and all that. *ACM Transactions on Programming Languages and Systems*, *6*(2), 281-296.
Livshits, B. (2006, December). *Improving software security with precise static and runtime analysis.* Thesis,
Lyon, G. (2006). *Top 100 network security tools* [Web page]. Retrieved April 20, 2009, from http://sectools.org/
Maidl, M., Gilmore, S., Haenel, V., & Kloul, L. (2005). Choreographing security and performance analysis for web services. In *Lecture notes in computer science.* (pp. 200-14). Springer Berlin / Heidelberg.
Manson, G., Mouratidis, H., & Giorgini, P. (2003). Integrating security and systems engineering: Towards the modelling of secure information systems. In *Lecture notes in computer science.* Springer Berlin / Heidelberg.
Maña, A., Montenegro, J. A., Rudolph, C., & Vivas, J. L. (2003). A business process-driven approach to security engineering. In *DEXA '03: Proceedings of the 14th international workshop on database and expert systems applications.* IEEE Computer Society.

Maña, A., Rudolph, C., Spanoudakis, G., Lotz, V., Massacci, F., Melideo, M., et al. (2007). Security engineering for ambient intelligence: A manifesto. In *Integrating security and software engineering: Advances and future vision.* (pp. 244-70).
Martin, M., Livshits, B., & Lam, M. (2005). Finding application errors and security flaws using PQL: A program query language. In *20Th annual ACM conference on objects-oriented programming, systems, languages and applications.*
Massacci, F., Mylopoulos, J., & Zannone, N. (2007). Computer-Aided support for secure tropos. *Automated Software Engg.*, *14*(3), 341-364.
Mathe, J., Duncavage, S., Werner, J., Malin, B., Ledeczi, A., & Sztipanovits, J. (2007). Implementing a model-based design environment for clinical information systems. In *Workshop on model-based trustworthy health information systems.*
Mell, P., Scarfone, K., & Romanosky, S. (2007). *CVSS: A complete guide to the common vulnerability scoring system.*
Méry, D., & Merz, S. (2007). Specification and refinement of access control. *Journal of Universal Computer Science*, *13*(8), 1073-1093.
MITRE (2007). *Open vulnerability and assessment language (OVAL).*
Moebius, N., Haneberg, D., Reif, W., & Schellhorn, G. (2007). A modeling framework for the development of provably secure e-commerce applications. In *ICSEA '07: Proceedings of the international conference on software engineering advances.* IEEE Computer Society.
Montangero, C., Buchholtz, M., Gilmore, S., & Haenel, V. (2005). End-To-End integrated security and performance analysis on the DEGAS choreographer platform. In *Lecture notes in computer science.* (pp. 286-301). Springer Berlin / Heidelberg.
Mylopoulos, J., Giorgini, P., & Massacci, F. (2003). Requirement engineering meets security: A case study on modelling secure electronic transactions by VISA and mastercard. In *Lecture notes in computer science.* (pp. 263-76). Springer Berlin / Heidelberg.
OASIS (2006, October 12). *Reference model for service oriented architecture V1.0.* Retrieved April 21, 2009, from http://docs.oasis-open.org/soa-rm/v1.0/soa-rm.pdf
Ou, X., Boyer, W., & McQueen, M. (2006). A scalable approach to attack graph generation. In *CCS '06: Proceedings of the 13th ACM conference on computer and communications security.* ACM.
Pauls, K., Kolarczyk, S., Koch, M., & Löhr, K. (2006). Sectool
– supporting requirements engineering for access control. In *Lecture notes in computer science.* (pp. 254-67). Springer Berlin / Heidelberg.
Piattini, M., & Fernández-Medina, E. (2004). Extending OCL for secure database development. In *Lecture notes in computer science.* (pp. 380-94). Springer Berlin / Heidelberg.
Pironti, A., & Sisto, R. (2008). Soundness conditions for message encoding abstractions in formal security protocol models. In *ARES '08: Proceedings of the 2008 third international conference on availability, reliability and security.* IEEE Computer Society.
Ray, I., France, R., Li, N., & Georg, G. (2004). An aspect-based approach to modeling access control concerns. *Information and Software Technology*, *46*(9), 575-587.
Redwine, S. (2007). *Introduction to modeling tools for software security.*
Reichert, M., Rinderle, S., Kreher, U., Acker, H., Lauer, M., & Dadam, P. (2006). ADEPT next generation process management technology — tool demonstration. In *Caise'06 forum.* Luxembourg.
Rosado, D. G., Fernandez-Medina, E., Piattini, M., & Gutierrez, C. (2006). A study of security architectural patterns. In *ARES '06: Proceedings of the first international conference on availability, reliability and security.* IEEE Computer Society.
Saltzer, J. H., & Schroeder, M. D. (1975). The protection of information in computer systems. In *IEEE, proceedings.*
Santen, T. (2006). Stepwise development of secure systems. In *Lecture notes in computer science.* (pp. 142-55). Springer Berlin / Heidelberg.
Santen, T., Heisel, M., & Pfitzmann, A. (2002). Confidentiality-Preserving refinement is compositional - sometimes. In *ESORICS '02: Proceedings of the 7th european symposium on research in computer security.* London, UK: Springer-Verlag.
Schneider, F. B. (1998). *Trust in cyberspace.* National Academy Press.
Schneier, B., & Kelsey (1999). Security audit logs to support computer forensics. *ACM Transactions on Information and System Security*, *2*(2), 159-176.

Seehusen, F., & Stølen, K. (2006). Information flow property preserving transformation of UML interaction diagrams. In *SACMAT '06: Proceedings of the eleventh ACM symposium on access control models and technologies.* ACM.
Shabalin, P., & Jürjens, J. (2004). Automated verification of UMLsec models for security requirements. In *Lecture notes in computer science.* (pp. 365-79). Springer Berlin / Heidelberg.
Sindre, G., & Opdahl, A. L. (2005). Eliciting security requirements with misuse cases. *Requir. Eng.*, *10*(1), 34-44.
Siveroni, I., Zisman, A., & Spanoudakis, G. (2008). Property specification and static verification of UML models. In *ARES '08: Proceedings of the 2008 third international conference on availability, reliability and security.* IEEE Computer Society.
Spanoudakis, G., Kloukinas, C., & Androutsopoulos, K. (2007). Towards security monitoring patterns. In *SAC '07: Proceedings of the 2007 ACM symposium on applied computing.* ACM.
Swiderski, F., & Snyder, W. (2004). *Threat modeling.* Microsoft Press Redmond, WA, USA.
Thompson, H. (2005). Application penetration testing. *IEEE Security and Privacy*, *3*(1), 66-69.
Turner, D. (2008). *Symantec global internet security threat report: Trends for july-december 2007.*
UMLsec group (2009). *Security analysis tool* [Web page]. Retrieved January 12, 2009, from http://www.umlsec.org
Whittle, J., Wijesekera, D., & Hartong, M. (2008). Executable misuse cases for modeling security concerns. In *ICSE '08: Proceedings of the 30th international conference on software engineering.* ACM.
Wolff, B., Brucker, A. D., & Doser, J. (2006). A model transformation semantics and analysis methodology for secureuml. In *Lecture notes in computer science.* (pp. 306-20). Springer Berlin / Heidelberg.
Woodside, M., Petriu, D. C., Petriu, D. B., Xu, J., Israr, T., Georg, G., et al. (2009). Performance analysis of security aspects by weaving scenarios extracted from UML models. *J. Syst. Softw.*, *82*(1), 56-74.
Yoshioka, N., Honiden, S., & Finkelstein, A. (2004). Security patterns: A method for constructing secure and efficient inter-company coordination systems. In *EDOC '04: Proceedings of the enterprise distributed object computing conference, eighth IEEE international.* IEEE Computer Society.
Yskout, K., Scandariato, R., Win, B. D., & Joosen, W. (2008). Transforming security requirements into architecture. In *ARES '08: Proceedings of the 2008 third international conference on availability, reliability and security.* IEEE Computer Society.
Yu, E., & Elahi, G. (2008). A goal oriented approach for modeling and analyzing security trade-offs. In *Lecture notes in computer science.* (pp. 375-90). Springer Berlin / Heidelberg.
Yu, W., Aravind, D., & Supthaweesuk, P. (2006). Software vulnerability analysis for web services software systems. In *Computers and communications, 2006 ISCC '06 proceedings 11th IEEE symposium on.*
Yu, Y., Jürjens, J., & Mylopoulos, J. (2008). Traceability for the maintenance of secure software. In *24Th IEEE international conference on software maintenance.*
Zhang, G., Baumeister, H., Koch, N., & Knapp, A. (2005). Aspect-Oriented modeling of access control in web applications. In *Proc. 6Th int. Workshop on aspect oriented modeling, AOSD.*

[i] http://www.metasploit.com/
[ii] Microsoft http://www.microsoft.com/technet/security/current.aspx
[iii] SecWatch http://secwatch.org/
[iv] http://www.sans.org/newsletters/risk/display.php
[v] Open Web Application Security Project http://www.owasp.org/
[vi] Milw0rm http://www.milw0rm.com/
[vii] SecuriTeam http://www.securiteam.com/
[viii] French Security Incident Response Team http://www.frsirt.com/english/

[ix] IBM Internet Security Systems X-Force. Alerts and Advisories, 2007. http://www.iss.net/rss.php
[x] National Institute of Standards and Technology. National Vulnerability Database NVD http://nvd.nist.gov
[xi] Common Vulnerabilities and Exposures http://cve.mitre.org/
[xii] Open Source Vulnerability Database OSVDB http://osvdb.org
[xiii] SecurityFocus. SecurityFocus Vulnerability Database, 2007. http://www.securityfocus.com/bid
[xiv] SecurityTracker http://securitytracker.com/
[xv] Oracle. http://www.oracle.com/technology/products/jdev/index.html
[xvi] The Eclipse Foundation. http://www.eclipse.org