Model-based Security Engineering: Managed Co-Evolution of Security Knowledge and Software Models [*]

Jens Bürger[1], Jan Jürjens[3], Thomas Ruhroth[1], Stefan Gärtner[2], and Kurt Schneider[2]

[1] Technische Universität Dortmund, Germany
{thomas.ruhroth,jens.buerger}@cs.tu-dortmund.de
[2] Leibniz Universität Hannover, Germany
{stefan.gaertner,kurt.schneider}@inf.uni-hannover.de
[3] Technische Universität Dortmund and Fraunhofer ISST, Germany
jan.jurjens@cs.tu-dortmund.de

Abstract. We explain UMLsec and associated techniques to incorporate security aspects in model-based development. Additionally, we show how UMLsec can be used in the context of software evolution. More precisely, we present the SecVolution approach which supports monitoring changes in external security knowledge sources (such as compliance regulations or security databases) in order to react to security related modification and to support the associated co-evolution of the UMLsec models.

1 Introduction

Security modeling allows one to consider security issues at an early stage in the development process [15]. Many security problems are induced by design flaws which leads to problems in the developed software. A software design which is augmented by security information helps the developer to avoid vulnerabilities in the software design.

Although several approaches for model-based secure software engineering exist, few of these include automated tools for formally verifying the models against the security requirements. Here, we focus on one such approach called *UMLsec* [26]. It extends the Unified Modeling Language (UML) and offers automated tools to verify UML models against security requirements [19] (cf. Fig. 1). The *UMLsec tool* as well as its successor *CARiSMA* [11] support the analysis of the security aspects expressed in the security extension UMLsec [26] (cf. Fig. 2). CARiSMA is a reimplementation of the UMLsec tool and build upon the Eclipse Modeling Framework (EMF). It thus supports UML and UMLsec models but is especially extensible to support further modeling languages (e.g. BPMN as

[*] Funded by the DFG project SecVolution (JU 2734/2-1, SCHN 1072/4-1), part of the priority programe SPP 1593 "Design For Future - Managed Software Evolution".

domain-specific language). Moreover, CARiSMA ist designed highly modular and offering a flexible plugin-structure. CARiSMA as well as the UMLsec tool mainly focus on the verification of the most important security requirements, which can be directly used in the model, together with their formal definitions.

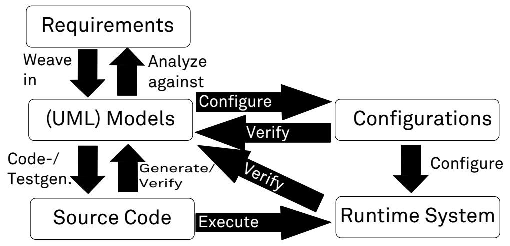

Fig. 1: Model-based Security Engineering

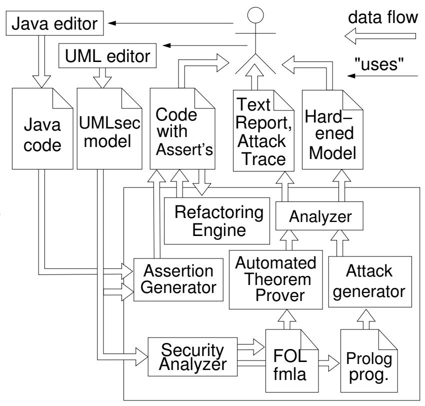

Fig. 2: UMLsec Tool Suite

Security Decay in Ageing Information Systems: Information systems are exposed to constantly changing environments which require constant updating. Software "ages" not by wearing out, but by failing to keep up-to-date with its environment [32]. New technology, changing customer requirements, and new knowledge on various software development issues require constant updating. An information system that does not react to changes in its environment will soon be outdated. This is especially true for security and secure development [12]. When an information system handles assets of a company or an organization, any security loophole can be exploited by attackers. Advances in knowledge and technology of attackers are part of the environment of a security-relevant information system. Outdated security precautions can, therefore, permit sudden and substantial losses [3]. Security in long-living information systems, thus, requires an on-going and systematic evolution of knowledge and software for its protection. Thus, techniques, tools and processes are desired to support security requirements and design analysis techniques for evolving information systems in order to ensure "lifelong" compliance to security requirements.

SecVolution We therefore introduce the SecVolution approach for demonstrating how to cope with (co-)evolution of UMLsec models. The ultimate goal of SecVolution is to preserve the security of an information system by adapting software models through the use of external and internal knowledge sources. As presented in the workflow in Fig. 3, our approach supports reusing security engineering knowledge gained during the development of security-critical software and feeding it back into the development process. The information flow interface

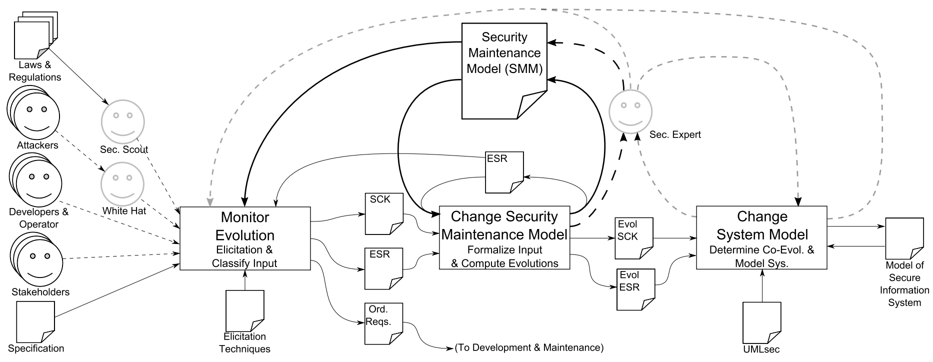
Laws & Regulations
Sec. Scout
Attackers
White Hat
Developers & Operator
Stakeholders
Specification
Monitor Evolution
Elicitation & Classify Input
Elicitation Techniques
Security Maintenance Model (SMM)
ESR
SCK
ESR
Ord. Reqs.
(To Development & Maintenance)
Change Security Maintenance Model
Formalize Input & Compute Evolutions
Sec. Expert
Evol SCK
Evol ESR
Change System Model
Determine Co-Evol. & Model Sys.
UMLsec
Model of Secure Information System

Fig. 3: Extended information flow model of the proposed *SecVolution* approach (ESR = Essential Security Requirements; SCK = Security Context Knowledge). The information flow syntax is described in [43,42].

of UMLsec is depicted on the right hand side. UMLsec supports the construction activity. The input of that activity is an improved requirements specification with security-relevant parts being identified and marked. The ultimate outcome is supposed to be a secure system, by making use of a set of security-enhanced UMLsec models.

Since changes in the environment are the reason for updating an information system, a knowledge model, namely the *Security Maintenance Model* (SMM), plays an intermediary role. Incoming information on changes is represented in that model. For this purpose, heuristic tools and techniques are used to support elicitation of relevant changes in the environment. Then, findings are formalized for semi-automatic security updates and stored in the SMM. Therefore, it is essential to stay aware of potential sources for relevant changes and to prepare for eliciting new knowledge. On the back-end, represented changes trigger respective changes in the software models and their security aspects by means of co-evolution. This leads to fast reactions and security updates in order to keep information systems secure and “young at heart”.

CoCoME Case Study: To illustrate the usage of UMLsec regarding secure information flow as well as (co-)evolution, we introduce the *Common Component Modeling Example (CoCoME)* as running case study. CoCoME represents a point-of-sale system as it can be found in most supermarkets. The system consists of a number of cash desk PCs connected to a store server in a hierarchical manner. A number of store servers again is connected to a central enterprise server. As the communication paths between these systems are used to transmit business as well as personal data (e.g. when processing EC transactions), communication between the systems has to satisfy given security requirements. Moreover, a lot of additional hardware is plugged into the cash desk PC providing various entry points to the whole trading system. For additional information and models of CoCoME, we refer to [18].

The remainder of this paper is organized as follows: In Sec. 2, we shortly recall some relevant background on the UMLsec approach. In Sec. 3, we explain how to model security knowledge and how to deal with its evolution. In Sec. 4, we explain how, in reaction to changes in the security knowledge, the security models can be co-evolved accordingly to deal with these changes appropriately. After a discussion of further reading in Sec. 5, we end with a conclusion.

2 Model-based Security Engineering using UMLsec

In the UML extension UMLsec [26], recurring security requirements (such as secrecy, integrity, authenticity and others) and security assumptions on the system environment can be specified within a UML specification as annotations. This way we can encapsulate knowledge on prudent security engineering 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.

More information about the UMLsec approach and its notation can be found in [26,25]. Some applications are reported in [29,23,28,24,21,20,27].

2.1 Secure Information Flow

Secure Information Flow (SIF) describes techniques to analyze and prevent the flow of confidential information from a trusted ("high") to an untrusted ("low") domain [17,35]. While dealing with SIF, a direct and an indirect flow can be distinguished. Figure 4 shows both of them. In the left part a boolean variable l in an untrusted environment is assigned a value from a high domain (h). Here,

```
l = h;
```

```
if (h)
   l = false;
else
   l = true;
fi
```

Fig. 4: Direct and indirect information flow: h is a variable containing confidential (high) data and l is a variable containing normal data (low) data. For simplicity both variables are booleans.

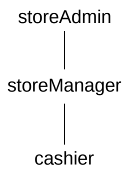

Fig. 5: Hasse-Digram for the security levels used in the running case study.

the confidential information flows directly to the untrusted domain. The right side shows an indirect information flow. Through the if statement the information in the low domain is influenced in a way that the high value can be simply computed. This easy example shows that secure information flow is not easily analyzed using dataflow in program, furthermore all side-effects from the program logic need to be considered. In the literature, therefore, an approach using observations of an program is often used. A program interacts with its environment and events can be observed. These events are categorized like the data into high and low data, which are used to describe the SIF properties in terms of program observations. One of these approaches, which covers many others, is the *Modular Assembly Kit for Security* (MAKS) [34,33]. It can be used to express secure information flow properties in a modular and uniform way. The fundamental elements are *Basic Security Predicates* (BSP). All BSPs are parameterized with a view $\mathcal{V}$, denoting the elements which are confidential. A MAKS *view* $\mathcal{V} = (V, N, C)$ [34] is a disjoint partition of the event set E into three sets V, N, C.

	visible	invisible
confidential	$\emptyset$	C
not confidential	V	N

The set C collects confidential events which should not be seen. The events of V are visible and hold non confidential data. Events which cannot be observed and are not confidential are collected in the set N. The combination of visible but confidential events makes no sense, because visibility and confidentiality are contradicting.

The system model is given as a prefix closed set Tr of all system traces where a trace is a sequence of events E. The system traces Tr are prefix closed if for all traces all prefixes are in Tr also. For example, if the sequence $\langle abc \rangle$ is in Tr then the traces $\langle ab \rangle$, $\langle a \rangle$ and $\langle \rangle$ need to be in Tr also.

Using a view $\mathcal{V}$ we can define BSPs. One example for such a BSP is *strict removal* (SR):

$$SR_{V,N,C}(Tr) := \forall \tau \in Tr : \tau|_{V \cup N} \in Tr$$

Strict removal describes that all "confidential" events C are independent of the "visible" V and "neither-nor" events N and thus no information about confidential events can be inferred from the others.

BSPs can be combined and parameterized by views. By using BSPs many SIF properties can be expressed, including many traditional notions. For example, non-interference ($NF_{H,L}(Tr) := \forall \tau \in Tr : \tau|_L \in Tr$) can be modeled using SR [34]:

Let L be a set of low events and H be a set of high events with $L \cup H = E$. Then the following equality holds:

$$SR_{H,\emptyset,L}(Tr) = NF_{H,L}(Tr)$$

In many applications the distinction in two security levels (high and low) is too simple. A common used approach using partially ordered sets (poset) can be used to reduce a multi-level problem. For example, a widely known system using this approach is the linear system of the NATO defining the security levels unclassified, classified, secret and top secret. In general, the security level structure is described by a *partially ordered set* $(A, \leq)$ with the set A containing the security levels and an partial order $\leq$ between the levels. If P denotes the set of all permissions, then a security level can be described as a subset of P. The family of security levels together with the operation subset $\subset$ builds a poset. This can be depicted as a Hasse Diagram (cf. Figure 5). Each connection from a higher node x to a lower node y means $x \leq y$. In security analysis, the poset approach can be analyzed in terms of the high/low-approach. For each security level S we define two sets H and L. H contains all permissions of S and all greater sets of the poset. The set L holds all other permissions. This translation is the input for high/low-analysis. If the analysis for all security levels fulfilled the poset holds the SIF property.

In this tutorial, the security levels are modeled as sets of events that are allowed to be seen by users holding this permission. Therefore, the relation between the security levels is modeled as a poset.

2.2 CoCoME Case Study: Modeling Secure Information Flow

In this section, we illustrate the use of UMLsec and the extension to its core to model secure information flow as introduced above. For this purpose, we extend particular models of the CoCoME system.

Figure 6 shows a class diagram that depicts the components and interfaces defined to handle inventory data of a specific store. Some details are omitted in the figure due to the sake of readability. We only show the elements that are relevant for the example. The operations defined as part of the interfaces model the information flow and data handling inside the store.

According to the requirements [18], it seems reasonable to define three roles and arrange their rights as a poset as part of the $\langle\langle DefSecurityLevels\rangle\rangle$ stereotype (cf. Figure 5). The lowest level is set to the *cashier* as he only needs limited access to the system and only uses few operations respectively. In fact, the cashier needs to book sales, query stock items and specific products. The next level is defined by the role *manager*. This role resembles the one of the store's manager. In consequence, the manager needs to access a number of administrative operations such as `queryLowStockItems`. The third role necessary to model the secure information flow of this example is the administrator who maintains the IT of the store. This role, called *storeAdmin*, is the only one to be able to set the store's ID of the shop system (`setStoreId`).

3 Modeling and Evolution of Security Knowledge

Challenges in the evolution of an information system are driven by unforeseeable changes in the environment of a security-relevant information system: New

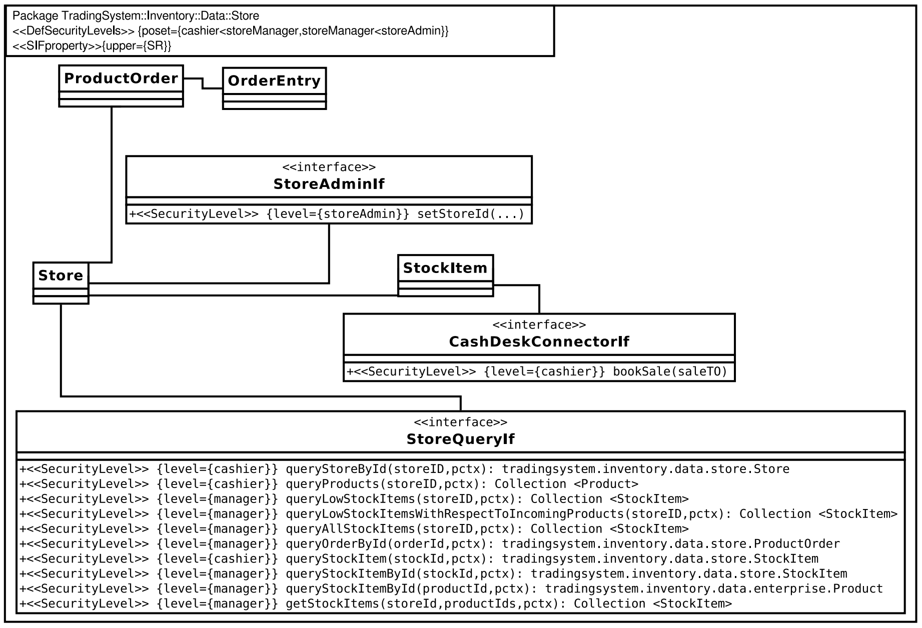
Package TradingSystem::Inventory::Data::Store
<<DefSecurityLevels>> {poset={cashier<storeManager,storeManager<storeAdmin}}
<<SIFproperty>>{upper={SR}}
ProductOrder
OrderEntry
<<interface>>
StoreAdminIf
+<<SecurityLevel>> {level={storeAdmin}} setStoreId(...)
Store
StockItem
<<interface>>
CashDeskConnectorIf
+<<SecurityLevel>> {level={cashier}} bookSale(saleTO)
<<interface>>
StoreQueryIf
+<<SecurityLevel>> {level={cashier}} queryStoreById(storeID,pctx): tradingsystem.inventory.data.store.Store
+<<SecurityLevel>> {level={cashier}} queryProducts(storeID,pctx): Collection <Product>
+<<SecurityLevel>> {level={manager}} queryLowStockItems(storeID,pctx): Collection <StockItem>
+<<SecurityLevel>> {level={manager}} queryLowStockItemsWithRespectToIncomingProducts(storeID,pctx): Collection <StockItem>
+<<SecurityLevel>> {level={manager}} queryAllStockItems(storeID,pctx): Collection <StockItem>
+<<SecurityLevel>> {level={manager}} queryOrderById(orderId,pctx): tradingsystem.inventory.data.store.ProductOrder
+<<SecurityLevel>> {level={cashier}} queryStockItem(stockId,pctx): tradingsystem.inventory.data.store.StockItem
+<<SecurityLevel>> {level={manager}} queryStockItemById(stockId,pctx): tradingsystem.inventory.data.store.StockItem
+<<SecurityLevel>> {level={manager}} queryStockItemById(productId,pctx): tradingsystem.inventory.data.enterprise.Product
+<<SecurityLevel>> {level={manager}} getStockItems(storeId,productIds,pctx): Collection <StockItem>

Fig. 6: Class diagram reflecting data handling of a single CoCoME store featuring UMLsec SIF

regulations require compliance; the source of new regulations (e.g. standards organizations, law-makers, interest groups etc.) are usually well-known. Moreover, failures will be detected in the construction of the information system itself. When it comes to new or changing technologies and knowledge, it is difficult to determine the impact on security of the information system. For example, faster computers can make it easier to decode simplistic encodings. New approaches like rainbow tables [37] can compromise password encoding schemes that were considered sufficient before. A rainbow table is a preprocessed table for reversing cryptographic hash functions. This technique, published in 2003, is used for cracking password hashes and has be successfully applied to various systems.

Knowledge management is an essential part of our SecVolution approach to cope with security issues properly. In this section, we therefore focus on organizing security knowledge and its evolution. Whenever security knowledge changes, software engineers need to know how to adapt the regarded information system, so that a certain level of security is retained.

3.1 Modeling Security Context Knowledge

Security knowledge generally contains essential axioms, concepts and their relations including all important hierarchies and constraints. Additionally, *Security Context Knowledge* (SCK) is defined as security-relevant knowledge of the environment of an information system, including but not limited to attacker types and their abilities, encryption protocols and their robustness against different attacks, etc. It is contained in information sources of various kind (e.g. security guidelines and obligations [9] or attack and vulnerability reports [47]). In SecVolution, security context knowledge is part of the *Security Maintenance Model* (SMM) as presented in Figure 3. Since this knowledge is not necessarily limited and cannot guarantee that we will discover complete information about security, the knowledge must be extended at any time.

In recent years, different ontologies and meta-models for security knowledge have been proposed. One can distinguish between asset-centric modeling, which focuses on the values that are threatened, and system-centric modeling, which depicts systems and their vulnerabilities. Since SecVolution should deal with versatile security knowledge on the one hand and an actual information system that is to be maintained on the other hand, we need an integrated view that incorporates security- and system-specific aspects.

In [36], a generic meta-model for IT security obtained from an extensive literature review is presented. It provides common security concepts and properties that are the same across different domains. It has an asset-centric view and distinguishes between the different properties of attacks and countermeasures on a conceptual level (e.g. is an attack taking place in public or in private, using legal or criminal means; or whether it is detected by audits or by prediction). Further relevant meta-models and ontologies are proposed in [7,48,14]. A widely accepted meta-model considering security-relevant system properties is introduced in [45]. Further publications about security knowledge modeling can be found in [16].

In this tutorial, we use a minimal knowledge structure to model SCK, which can be found in each of the abovementioned meta-models and ontologies in one or the other way. It is shown in Figure 7.

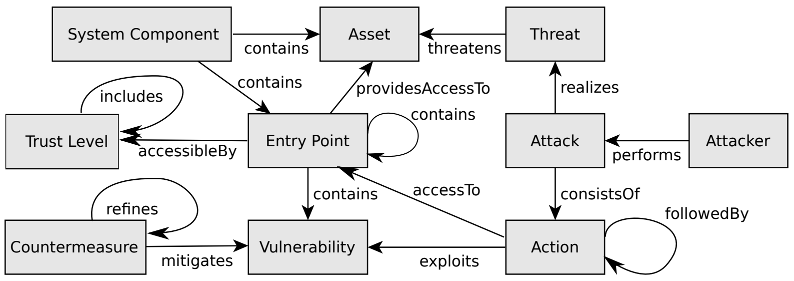

Fig. 7: Minimal core of our knowledge structure. [16]

To model an actual system from a security perspective, following concepts are defined. An *asset* is an item of interest worth being protected. This generally includes hardware (e.g. server, switches) and software (e.g. critical applications, services) components. Regarding information systems, assets are also sensitive or confidential information (e.g. passwords, user data, financial data and reports). To provide access to assets, an *entry point* is defined as the interface to interact with the system (e.g. login website, email, input field). Thus, each asset has at

Class	Subclasses
Trust Level	Agent (Person Affected, Third Party, Recipient)
Asset	Data (Personal Data)
Action	Anonymization, Data Usage, Data Processing, Pseudonymization, Approval, Notification, Allowed Action, Critical Action
System Component	Data Storage Device (File, Database), Data Processing Unit
Countermeasure	Access Control, Physical Access Control, Encryption, Availability Check

Table 1: Extension of the used knowledge structure to model privacy

least one entry point. A *trust level* describes which role has access to an asset using a specific entry point (e.g. customer, manager, administrator). *System components* model the regarded information system focusing on assets and entry points.

To model security knowledge from an attacker perspective, the following concepts are defined. A *threat* is the possibility to perform a successful attack on a specific asset (e.g. execute unauthorized code or commands, expose sensitive data). Here, an *attack* is a sequence of malicious *actions* that are performed by an attacker (e.g. cross-site scripting, denial-of-service attack). To perform a successful attack, the attacker uses vulnerable entry points. Here, a *vulnerability* is a system property that violates an explicit or implicit security policy (e.g. improper neutralization of input, missing encryption of sensitive data). Thus, it facilitates unintended access or modification of assets. To mitigate a certain threat, *countermeasure*s are used to fix the respective vulnerability (e.g. input validation, encryption of sensitive data).

Note that this knowledge structure is not limited to the concepts depicted here. It can be extended to fulfill further domain-specific requirements and constraints. For example, if we need to model the German privacy directive as declared in [10], several concepts must be extended as listed in Table 1. To use the knowledge structure for an actual system, system components, assets, entry points and their relationships must be identified. This can be done by using threat modeling as explained in [45].

To realize our knowledge base in practice, we decided to use the Web Ontology Language (OWL) [38]. The SCK is therefore modeled in terms of *concepts* and *individuals*. A concept is a collection of individuals, which share some properties. Individuals are the basic elements of an ontology that describe the actual knowledge. One advantage is that we are able to use standard OWL tools such as Protégé [39] to edit knowledge elements. Moreover, reasoning and query languages exists and allow to have unified access to knowledge. To extend the knowledge structure as mentioned, OWL provides a mechanism to import knowledge elements from other ontologies.

3.2 Managing Evolving Security Knowledge

Security knowledge must be up-to-date in order to preserve security sustainably. The problem is to determine which security information remains valid, which information changes, how it changes, and why it changes. Moreover, information systems are complex due to various technologies and frameworks used for implementation.

To cope with security issues in long-living information systems, software engineers need to overview numerous knowledge sources during maintenance. As a consequence, they need methods to support identification and analysis of security loopholes and design flaws in development artifacts based on security-relevant knowledge. In particular, the evaluation whether a vulnerability can be exploited by an attack may alter with each change of the system itself or of the system context represented in the SMM. For example, encryption of a data connection might become vulnerable due to change in configuration or due to the development of more powerful algorithms for cracking passwords. To describe evolution of SCK, knowledge elements can be added, modified and deleted (atomic edit operations). If a change is reported by a knowledge source, the information must be classified in either ordinary requirements, essential security requirements, or security context knowledge.

SecVolution is not intended to detect or even to forecast new security issues or to mitigate known exploits automatically. It is rather meant as a methodology to systematically describe and share security knowledge and to make it usable for non-security experts. For this purpose, we developed heuristic tools and techniques to support security analysis of development artifacts and monitoring of relevant changes in the environment. In [16], a security assessment of natural-language requirements based on heuristics and reported security incidents is proposed.

3.3 CoCoME Case Study: Managing Security Knowledge

In this section, we illustrate the use of our knowledge structure to model security problems. Regarding the CoCoME case study, we consider an extension of the trading system which enables customers to order goods online and to come to the shop to pick their order up (pick-up shop). The pick-up shop makes use of hidden fields to pass values from one page-call to another. This simplifies the server component, since it can be implemented stateless. Stateless in this case means that every HTTP request happens in isolation. Thus, each HTTP request must include all information necessary for the server to process the request successfully. By doing so, the server does not need to store the data and neither to cope with many special cases like session aborts. Statelessness also corresponds to the resource-oriented architecture of the web.

In this example, we assume that an attacker exploited hidden fields to change the price of an order in our pick-up shop. Here, the price is stored in a hidden field to allow calculations, like the total over all items. It is forwarded using a HTTP request to submit the order. On the server side, the integrity of the hidden values is not checked explicitly. Therefore, the prices in the request can be altered and used in the subsequent delivery process.

A similar vulnerability has been reported as CVE-2000-1001 [46] in the Common Vulnerabilities and Exposures (CVE) database [47]. The CVE database provides security information about known vulnerabilities and exposures. The goal is to standardize the identification of publicly known vulnerabilities. The CVE database serves as a reference by software engineers for identifying and mitigating known vulnerabilities.

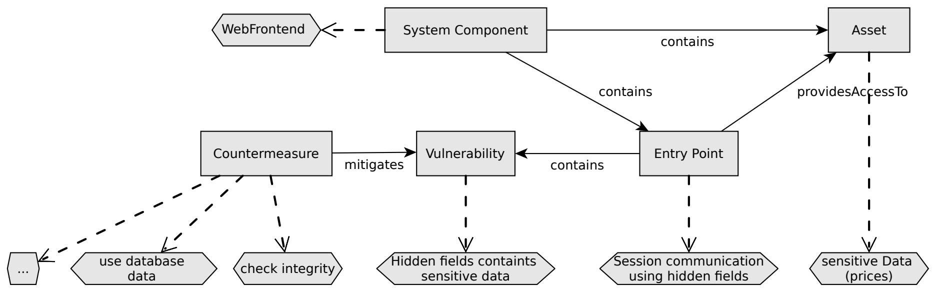

Fig. 8: Elements that are added to the SCK to reflect knowledge about a new attack and countermeasures

The knowledge about the vulnerability, the attack, and its entry points as introduced in the preceding section needs to be added into the SMM. For this, we need to modify the SCK. In this case, we add the notion of sensible data (e.g. the price in the attack), the skills of the attacker and the countermeasures for the attack. Possible countermeasures include using the price of the product in the price database and not using the price in the HTTP request, or the integrity check of the HTTP request that is transmitted. For simplicity, we show only the addition of the countermeasure integrity check of the HTTP request. In Figure 8, we see a fraction of a SCK that is already extended by new countermeasures. The system component affected is the web front-end which exposes an entry point by the definition and use of hidden fields for state communication between the HTTP requests. When these hidden fields contain sensible data (*asset*) this leads to the corresponding vulnerability.

4 Co-Evolving Security Models

The ongoing change through modified requirements, corrections of discovered problems etc. is often referenced as software evolution. A change occurs when certain facts that are true in a situation are no longer true in a later situation [44]. Additionally, software evolution describes the process in the maintenance phase which precedes the servicing and decommission phase. Since the evolution should not depend on ad-hoc change, we also need a notion of evolution in the area of software and security knowledge modeling. Here, evolution is often characterized as ongoing change to the model.

In the remainder we show how knowledge evolution affects co-evolution of security models. Moreover, we present an approach to model co-evolution of UMLsec models based on security knowledge stored in the SMM.

4.1 Evolution of Knowledge and Co-Evolution of Security Models

If the information system was developed in a model-driven way, the adaptations with respect to changed security knowledge or requirements can even be automatized with co-evolution expressed as model transformations. For this purpose, adaption information describing potential changes and how to deal with such changes if they appear must also be stored in the SMM.

Since knowledge evolution and the corresponding co-evolution of the system model are manifold, we designed the SMM consisting of three parts. Firstly, the *Security Context Knowledge* (SCK) incorporates security-relevant knowledge of the environment of the regarded system. Secondly, the *Catalog of Reactions* (CoRe) bridges the gap between changes in the knowledge (i.e. evolutions of the SCK) and consequent changes that have to be applied to the system model (i.e. co-evolution). Thirdly, the *History of Evolutions* (HoE) is a supportive element that is used to determine changes happened to the SMM so far.

When knowledge sources are identified, information on new knowledge and its potential impact on security must be elicited. This analysis needs to reach a level of detail that allows conceiving countermeasures and updating of models during co-evolution. Therefore, each change in the elicited knowledge needs to be reflected in the models, such that the security knowledge and the models evolve together. Thus, a side by side evolution is referenced to as co-evolution.

Figure 9 shows the interrelationship between knowledge evolution and co-evolution of the system model. First, a security analysis evaluates whether the system model is secure in terms of the knowledge represented in the SMM at design-time. This can be achieved by using our tool-support CARiSMA[11]. Changes in the SCK may trigger respective changes in the system model and their security aspects as described by the CoRe. Evolution of the SCK con-

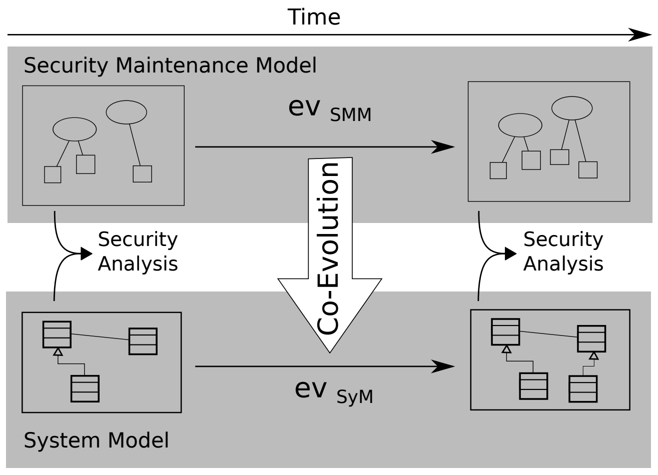

Fig. 9: Interrelationship between security knowledge evolution and co-evolution of the system model built upon this knowledge

tained in the SMM is described in terms of atomic evolution steps. Depending on the parameters of all evolutions that took place, it has to be checked whether security is retained. Unfortunately, the changes to the SMM cannot be fully automatically used to adapt the system model, since these changes require design decisions as well as that they depend on the language of the model. Thus, we use a flexible approach that allows explicit modeling of different adaption options.

4.2 Modeling Security Maintenance Rules

To adapt the system model sufficiently, a sequence of evolution operations on the system model is necessary. The pairing of the evolution of two different artifacts (here SMM and system model) is referred to as co-evolution. These co-evolutions are derived from the sequence of applied edit operations which have been used to change the SCK. This interrelationship is modeled in the SMM using the *Security Maintenance Rules* (SMR). They are a part of the CoRe and build the connecting link between ev_{SMM} and possible corrective actions that have to take place at the system model. Since these rules depend on the security information in the SMM and are themselves part of the SMM, they can also evolve. The SMRs are to trigger reactions upon the security knowledge that has changed. Regarding the kinds of possible reaction types, an appropiate reaction can be the direct manipulation of the *System Model* (SyM). This can for example be achieved by making use of graph transformation techniques. Moreover, displaying instructions to the maintainer is also an important kind of reaction to security knowledge evolution. For example, a newly discovered attack that can be performed if the passwords used in the system underrun a certain length does not require changes at the model level. In fact, the countermeasure is to instruct the maintainer to ensure that all passwords used in the systems exceed a (new) minimum length. We define SMRs to describe the adaption opportunities for the SyM as a quadruple

$$(ev_{\mathrm{SMM}}, ev_{\mathrm{SyM}}, pars, pre). \quad (1)$$

The first evolution ev_{SMM} describes the changes to the SMM and the second evolution ev_{SyM} describes the corresponding changes that shall take place at the system model. Ev_{SMM} and ev_{SyM} are captured in a transformation-based manner. The predicate *pars* contains parameters that may additionally characterize both evolutions. Moreover, *pre* resembles a precondition that has to hold for the respective SMR to be applied. In summary, the latter two elements of an SMR ensure that the parameters of the evolutions are compatible to each other. Information about parameterizing ev_{SMM} is especially helpful because it can support parameterizing of more generic co-evolution functions. Seeing the relation of the parameters of the different evolutions as predicated allows us to deal with directly computed parameters and manually chosen parameters.

As discussed above, SMRs can also evolve. Moreover, it seems reasonable to not have any duplicates and detect variants of the same SMRs. We thus do not let SMRs be created or modified directly but provide editing operations. Using editing operations, we can attach operations that check certain conditions prior to modifying the set of SMRs. For example, adding a new countermeasure can be performed by an operation formally denoted as

$$add.countermeasure.type(countermeasure, vulnerability, pars, pre)$$

where the *countermeasure* is the countermeasure and *vulnerability* the vulnerability to be mitigated. *pars* and *pre* are used to parameterize the countermeasure and define a precondition prior to applying the corrective actions. *type* denotes the reaction's type. For example, *instruction*s to the maintainer or requesting a (sequence of) model alterations through graph $transformation$ steps are possible values.

It is difficult to describe evolutions for a general system model. Thus, the second part of the rules is described in terms of the regarded model language. Here, we choose UMLsec as introduced in Section 2, because it provides lightweight annotations of security properties and requirements. These annotations are used to ensure the security in the code.

The mapping between the SMM and the system model need not to be one-to-one. For each SMM evolution, we can define an arbitrary number of system model co-evolutions. In this case, the maintainer can choose between different solutions. This allows the modeling of different solutions for one given adaption. For example, the integrity of data communicated can be obtained by different adaptions. One possibility is to use digital signatures over the value. Another possibility is to check the value when it is communicated back to the issuer.

After describing the parts of the SMM including SCK and SMRs, we show an application of our approach by continuing our case study in the next section.

4.3 CoCoME Case Study: Adapting UMLsec Models

In this section, we illustrate the use of security maintenance rules to adapt UMLsec models. For this purpose, we consider the pick-up shop extension as introduced in Section 3.3. We use UMLsec to annotate security related annotations into the models. In detail, the security requirement of data integrity as resembled by the stereotype ⟨⟨integrity⟩⟩ .

At design-time, there is no integrity check of the hidden fields data transmitted between browser and the server component. Later, the analysis of the reported vulnerability leads to new knowledge about the use of hidden fields in a web page and data in our pick-up shop. In this example, this leads to the perception that the integrity of sensitive data needs to be checked and the prices need to be treated as sensitive data. Therefore, respective model elements need to be annotated with appropriate UMLsec stereotypes (e.g. ⟨⟨integrity⟩⟩) to enforce respective security checks as part of the implementation.

The compliance of security requirements annotated using UMLsec can be easily checked using our tools support. In the following, we focus on the integrity of sensitive data in hidden fields as introduced in Section 3.3. Here, we explain how the modified knowledge facilitates changes to the model such that the security of the information system under consideration is preserved.

Modeling Security Maintenance Rules As explained in Section 3.3, the SCK must be updated to reflect the new knowledge. After that, it is necessary to model SMRs that are used to react upon security issues. On this account, we need to define SMRs for every possible evolution of the SCK. Regarding the reported vulnerability, the countermeasure *integrity check* needs to be described as an evolution of a software model. Because an integrity check can be directly annotated in UMLsec, this rule mainly has to describe the location where to add the tag ⟨⟨integrity⟩⟩ and some parameters (what to check and where the integrity should be ensured). There are often several possible modifications, thus more co-evolutions can be added (see [22]). For example, the operation call following the formalization as introduced above to add a SMR for this countermeasure is as follows:

$$add.countermeasure.transformation(\texttt{"}\text{add integrity check}\texttt{"}, \texttt{"}\text{hidden fields manipulation}\texttt{"}, pars, pre)$$

with

$$pre = \emptyset$$
$$pars = (\text{fields} = \text{getFieldsMarked}(\texttt{"}\text{integrity}\texttt{"})) \tag{2}$$

For simplicity, we assume an auxiliary function to query the model. We use `getFieldsMarked` to get all fields which need to be secured by the SyM evolution. such functions can be realized, for example by using existing tool frameworks (e.g. CARiSMA).

Modifying the model After the SCK is updated and SMRs are added (ev_{SMM}), the co-evolution (ev_{SyM}) as introduced in Figure 9 takes place. Here, the SMR chosen to be applied is the one created through (2) and realizes the countermeasure *check integrity*. It thus leads to application of model transformations of the models that for example realize the addition of ⟨⟨integrity⟩⟩ stereotype. In

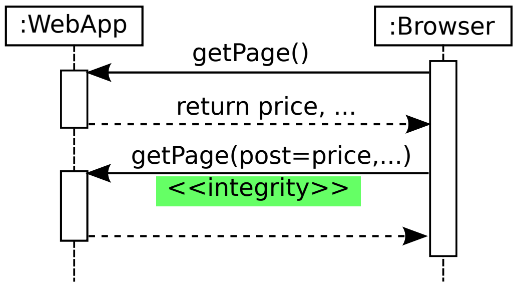

Fig. 10: One of the model changes modeling the use of the new countermeasure *check integrity* in the model. Here, the state-chart communication is annotated requiring that the integrity of the communicated data is checked. For simplicity the parameter tags of the stereotype are omitted.

summary, the resulting model is annotated with new security requirements. For example, the *check integrity* countermeasure leads to the requirement of checking the integrity of some post variables (see Figure 10).

5 Further Reading

Our approach combines methods and concepts from the field of secure software development as well as information security management. In the next subsection we give some further literature to other or connected approaches.

5.1 Model-based Security Engineering using UMLsec

Today, several approaches for model-based secure software engineering exist: Ray et al. [41] 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 delivers a system model that addresses access control concerns. Basin et al. [5] 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. Brose et al. [8] 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. The SECTET framework for Model Driven Security as proposed by Alam et al. [1] is applied for example towards a domain-specific approach for health care scenarios, including the modeling of access control policies, a target architecture for their enforcement, and model-to-code transformations. In the above-mentioned work, extensive security expertise is required to implement the proposed methods or to operate relevant tools. Moreover, several approaches are focused on a specific aspect of software systems such as access control.

5.2 Modeling and Evolution of Security Knowledge

A significant amount of work has been carried out on security knowledge management for information system. Belsis and Kokolakis [6] state that successful security knowledge management of information systems largely depends on the involvement of various stakeholders in security analysis, design, and implementation. However, they also found out in their field research that most stakeholders lack relevant security knowledge. To overcome this knowledge gap, Raskin et al. [40] present an ontology-driven security approach. Ontology organizes security knowledge (e.g. attacks and countermeasures, etc.) retrieved from natural language data sources. This approach can be seen as an interface of natural language processing and information security. Eloff and Solms [13] present a hierarchical framework for information security management. It is an early attempt to clarify security terminology and its interrelationships. The result is used to build up a hierarchical framework made available to the information system industry. Tsoumas and Gritzalis [48] suggest a security management approach for information systems which builds upon security knowledge gathered from various information sources. Security knowledge is organized in an ontology based on an extension of the DMTF Common Information Model. Other approaches dealing with similar ontologies to manage security knowledge properly are presented in [30,6,4]. In this approach the ontology is set up beforehand, considering widely-accepted standards and information about the infrastructure of the information system as well as established policy documents. Kritzinger and Smith [31] present a conceptual view of an information security retrieval and awareness model. The purpose of the model is to increase security awareness among employees of an organization. It ensures that technical aspects of information security do not outweigh human-related issues. Moreover, the presented model considers measuring and monitoring the current level of security awareness of each stakeholder in an organization. The measurement is mainly based on an awareness test consisting of multiple choice questions. The approach aims to link high-level policy statements and deployable security controls to support security expert's work. AlHogail and Berri [2] propose the development of an architecture sustaining security knowledge within an organization. The architecture aims to manage tailored security processes, policies and solutions. The goal of this approach is to capture and share security-related knowledge in order to efficiently react on security incidents and decrease dependencies on security experts. To capture security incidents as well as actions taken to mitigate the issue, a pre-defined report template is used. Subsequently, reports are analyzed manually to establish rules that are stored in an organization-wide knowledge base.

6 Conclusion

We described how to use UMLsec and the SecVolution approach for the security maintenance of evolving systems. UMLsec can be used to include different aspects of security requirements and techniques into model driven development processes build on UML. We demonstrated this by showing how to model secure information flow properties in a webshop case study build on CoCoME. We then showed how to model the evolution of security knowledge. We used an ontology to store security knowledge related to webstores. Based on this we can use the security knowledge to co-evolve the UMLsec model when the security knowledge changes.

References

1. M. Alam, M. Hafner, and R. Breu. Model-Driven Security Engineering for Trust Management in SECTET. *Journal of Software*, 2(1), February 2007.

2. Areej AlHogail and Jawad Berri. Enhancing it security in organizations through knowledge management. In *Information Technology and e-Services (ICITeS), 2012 International Conference on*, pages 1–6. IEEE, 2012.
3. Ross J. Anderson. *Security engineering - a guide to building dependable distributed systems (2. ed.).* Wiley, 2008.
4. Nicolas Anquetil, Káthia M. de Oliveira, Kleiber D. de Sousa, and Márcio G. Batista Dias. Software maintenance seen as a knowledge management issue. *Information and Software Technology*, 49(5):515–529, May 2007.
5. D.A. Basin, J. Doser, and T. Lodderstedt. Model driven security: From UML models to access control infrastructures. *ACM Trans. Softw. Eng. Methodol.*, 15(1):39–91, 2006.
6. Petros Belsis, Spyros Kokolakis, and Evangelos Kiountouzis. Information systems security from a knowledge management perspective. *Information Management & Computer Security*, 13(3):189–202, 2005.
7. Carlos Blanco, Joaquin Lasheras, Rafael Valencia-Garc, Eduardo Fern, Ambrosio Toval, and Mario Piattini. A Systematic Review and Comparison of Security Ontologies. *2008 Third International Conference on Availability, Reliability and Security*, 1(1):813–820, March 2008.
8. G. Brose, M. Koch, and K.-P. Löhr. Integrating Access Control Design into the Software Development Process. In *Integrated Design and Process Technology (IDPT)*, 2002.
9. Bundesamt für Sicherheit in der Informationstechnik (BSI). IT-Grundschutz-catalogues, 2013. `https://www.bsi.bund.de/EN/Topics/ITGrundschutz/ITGrundschutzCatalogues/itgrundschutzcatalogues_node.html`.
10. Bundesministerium des Inneren. Bundesdatenschutzgesetz. Bundesgesetzblatt. `http://www.bfdi.bund.de/DE/GesetzeUndRechtsprechung/BDSG/BDSG_node.html`.
11. CARiSMA project homepage. `http://carisma.umlsec.de/`.
12. Gurpreet Dhillon and Gholamreza Torkzadeh. Value-focused assessment of information system security in organizations. *Information Systems Journal*, 16(3):293–314, 2006.
13. M.M Eloff and S.H von Solms. Information Security Management: A Hierarchical Framework for Various Approaches. *Computers & Security*, 19(3):243–256, March 2000.
14. Stefan Fenz and Andreas Ekelhart. Formalizing information security knowledge. In *Proceedings of the 4th International Symposium on Information, Computer, and Communications Security (ASIACCS)*, page 183, New York, New York, USA, 2009. ACM Press.
15. Eduardo Fernández-Medina, Jan Jürjens, Juan Trujillo, and Sushil Jajodia. Model-driven development for secure information systems. *Information & Software Technology*, 51(5):809–814, 2009.
16. Stefan Gärtner, Thomas Ruhroth, Jens Bürger, Kurt Schneider, and Jan Jürjens. Maintaining Requirements for Long-Living Software Systems by Incorporating Security Knowledge. In *Proc. of the 22nd International Conference on Requirement Engineering*, 2014.
17. John Graham-Cumming. Some laws of non-interference (CSP algebra). In *Computer Security Foundations Workshop*, pages 22–33. IEEE Computer Society Press, 1992.
18. Sebastian Herold, Holger Klus, Yannick Welsch, Constanze Deiters, Andreas Rausch, Ralf Reussner, Klaus Krogmann, Heiko Koziolek, Raffaela Mirandola, Ben-

jamin Hummel, Michael Meisinger, and Christian Pfaller. CoCoME. In *The Common Component Modeling Example*, volume 5153 of *Lecture Notes in Computer Science*, pages 16–53, 2007.

19. S. Höhn and J. Jürjens. Rubacon: Automated support for model-based compliance engineering. In *International Conference on Software Engineering (ICSE)*, pages 875–878. ACM, 2008.
20. S.H. Houmb, G. Georg, J. Jürjens, and R.B. France. An integrated approach to security verification and security solution design trade-off analysis. In H. Mouratidis, editor, *Integrating Security and Software Engineering: Advances and Future Vision*, pages 190–219. Idea Group, August 2006. Invited chapter.
21. Siv Hilde Houmb, Geri Georg, Robert B. France, James M. Bieman, and Jan Jürjens. Cost-benefit trade-off analysis using BBN for aspect-oriented risk-driven development. In *10th International Conference on Engineering of Complex Computer Systems (ICECCS 2005), 16-20 June 2005, Shanghai, China*, pages 195–204. IEEE Computer Society, 2005.
22. Karthick Jayaraman and Grzegorz Lewandowski. Enforcing request integrity in web applications. *Data and Applications Security*, 14:225–240, 2010.
23. J. Jürjens. Secure information flow for concurrent processes. In C. Palamidessi, editor, *CONCUR 2000 (11th International Conference on Concurrency Theory)*, volume 1877 of *Lecture Notes in Computer Science*, pages 395–409. Springer-Verlag, 2000.
24. J. Jürjens. Modelling audit security for smart-card payment schemes with UMLsec. In M. Dupuy and P. Paradinas, editors, *Trusted Information: The New Decade Challenge*, pages 93–108. International Federation for Information Processing (IFIP), Kluwer Academic Publishers, 2001. Proceedings of the *16th International Conference on Information Security (SEC 2001)*.
25. J. Jürjens. Model-based security engineering with UML. In A. Aldini, R. Gorrieri, and F. Martinelli, editors, *Foundations of Security Analysis and Desing III: FOSAD 2004/2005 Tutorial Lectures*, volume 3655 of *Lecture Notes in Computer Science*, pages 42–77, 2005.
26. J. Jürjens. *Secure Systems Development with UML*. Springer, 2005.
27. J. Jürjens. Model-based security testing using UMLsec. *Electronic Notes in Theoretical Computer Science*, 220(1):93–104, December 2008.
28. J. Jürjens and G. Wimmel. Formally testing fail-safety of electronic purse protocols. In *16th International Conference on Automated Software Engineering (ASE 2001)*, pages 408–411. IEEE Computer Society, 2001.
29. J. Jürjens and G. Wimmel. Security modelling for electronic commerce: The Common Electronic Purse Specifications. In B. Schmid, K. Stanoevska-Slabeva, and V. Tschammer, editors, *Towards the E-Society: E-Commerce, E-Business, and E-Government*, pages 489–506. International Federation for Information Processing (IFIP), Kluwer Academic Publishers, 2001. First IFIP Conference on E-Commerce, E-Business, and E-Government (I3E 2001).
30. S Kesh and P Ratnasingam. A knowledge architecture for IT security. *Communications of the ACM*, 50(7), 2007.
31. E. Kritzinger and E. Smith. Information security management: An information security retrieval and awareness model for industry. *Computers & Security*, 27(5-6):224–231, October 2008.
32. M.M. Lehman. Programs, life cycles, and laws of software evolution. *Proceedings of the IEEE*, 68(9):1060–1076, Sept 1980.

33. H. Mantel. Possibilistic definitions of security – an assembly kit. In *Proceedings of the IEEE Computer Security Foundations Workshop*, pages 185–199, Cambridge, UK, July 3–5 2000. IEEE Computer Society.
34. H. Mantel. *A Uniform Framework for the Formal Specification and Verification of Secure Information Flow.* PhD thesis, Saarland University, Saarbrücken, Germany, 2003.
35. D. McCullough. Noninterference and the composability of security properties. In *IEEE Symposium on Security and Privacy*, pages 177 –186, apr 1988.
36. André Miede, Nedislav Nedyalkov, Christian Gottron, André König, Nicolas Repp, and Ralf Steinmetz. A Generic Metamodel for IT Security Attack Modeling for Distributed Systems. *2010 International Conference on Availability, Reliability and Security (ARES)*, pages 430–437, 2010.
37. Philippe Oechslin. Making a faster cryptanalytic time-memory trade-off. In Dan Boneh, editor, *Advances in Cryptology - CRYPTO 2003*, volume 2729 of *Lecture Notes in Computer Science*, pages 617–630. Springer Berlin Heidelberg, 2003.
38. W3C OWL Working Group. *OWL 2 Web Ontology Language: Document Overview.* W3C Recommendation, 27 October 2009. Available at `http://www.w3.org/TR/owl2-overview/`.
39. Protégé project homepage. `http://protege.stanford.edu/`.
40. Victor Raskin, Christian F. Hempelmann, Katrina E. Triezenberg, and Sergei Nirenburg. Ontology in information security: a useful theoretical foundation and methodological tool. In *Proceedings of the 2001 workshop on New security paradigms*, NSPW '01, pages 53–59, New York, NY, USA, 2001. ACM.
41. I. Ray, R.B. France, Na Li, and G. Georg. An aspect-based approach to modeling access control concerns. *Information & Software Technology*, 46(9):575–587, 2004.
42. Kurt Schneider, Eric Knauss, Siv Houmb, Shareeful Islam, and Jan Jürjens. Enhancing Security Requirements Engineering by Organizational Learning. *Requirements Engineering Journal (REJ), special issue on REFSQ'12*, 2012.
43. Kurt Schneider, Kai Stapel, and Eric Knauss. Beyond Documents: Visualizing Informal Communication. In *Proceedings of Third International Workshop on Requirements Engineering Visualization (REV '08)*, Barcelona, Spain, November 2008.
44. John F. Sowa. *Knowledge representation: logical, philosophical, and computational foundations*, volume 13. MIT Press, 2000.
45. Frank Swiderski and Window Snyder. *Threat Modeling.* Microsoft Press Corp., 2004.
46. The MITRE Corporation. Vulnerability Summary for CVE-2000-1001, 2001.
47. The MITRE Corporation. Common Vulnerabilities and Exposures, 2013.
48. B. Tsoumas and D. Gritzalis. Towards an Ontology-based Security Management. In *Proceedings of the 20th International Conference on Advanced Information Networking and Applications (AINA)*, volume 1, pages 985–992. IEEE, 2006.