The Journal of Systems & Software xxx (2018) xxx-xxx

Author preprint version

The Journal of Systems & Software

journal homepage: www.elsevier.com

A framework for semi-automated co-evolution of security knowledge and system models

Jens Bürger*, [a], Daniel Strüber[a], Stefan Gärtner[b], Thomas Ruhroth[a], Jan Jürjens[a, d], Kurt Schneider[c]

[a] University of Koblenz-Landau, Universitätsstraße 1, 56070 Koblenz, Germany
[b] adesso AG, Stockholmer Allee 200, 44269 Dortmund, Germany
[c] Lebniz University Hannover, Welfengarten 1, 30167 Hannover, Germany
[d] Fraunhofer ISST, Emil-Figge-Straße 91, 44227 Dortmund, Germany

ARTICLE INFO

Article history:
Received 1 April 2017
Received in revised form 20 December 2017
Accepted 5 February 2018
Available online xxx

Keywords:
Security requirements
Software evolution
Co-evolution
Software design
Security impact analysis

ABSTRACT

Security is an important and challenging quality aspect of software-intensive systems, becoming even more demanding regarding long-living systems. Novel attacks and changing laws lead to security issues that did not necessarily rise from a flawed initial design, but also when the system fails to keep up with a changing environment. Thus, security requires maintenance throughout the operation phase. Ongoing adaptations in response to changed security knowledge are inevitable. A necessary prerequisite for such adaptations is a good understanding of the security-relevant parts of the system and the security knowledge.

We present a model-based framework for supporting the maintenance of security during the long-term evolution of a software system. It uses ontologies to manage the system-specific and the security knowledge. With model queries, graph transformation and differencing techniques, knowledge changes are analyzed and the system model is adapted. We introduce the novel concept of *Security Maintenance Rules* to couple the evolution of security knowledge with co-evolutions of the system model.

As evaluation, community knowledge about vulnerabilities is used (Common Weakness Enumeration database). We show the applicability of the framework to the *iTrust* system from the medical care domain and hence show the benefits of supporting co-evolution for maintaining secure systems.

1. Introduction

Security and privacy are critical success factors for software-intensive systems. Security flaws and data breaches may impair customer satisfaction and sales revenues, while entailing a high cost for flaw repair and compensations. Even if a system is built with great efforts to shield against the security[1] threats known at the time of its initial deployment, a challenging situation arises when the system is to be maintained for an extensive lifetime. Such *long-living systems* are particularly prone to security issues, since assumptions and design decisions made during their development may be invalidated when the environment changes: due to newly discovered attack types, increasingly ambitious security laws, and continuously evolving stakeholder requirements, employed security mechanisms may become obsolete.

Therefore, an achieved level of security must be *actively maintained* during the long-term evolution of a system.

For these reasons, it is crucial to support developers during the *detection* and *repair* of security flaws after the environment changes. It is desirable to enable automated support for these tasks as far as possible, relieving developers from unnecessary burden, while involving them whenever their input is necessary. From this goal, three main challenges arise. First, the automated detection of security flaws requires to leverage available knowledge on security threats. This knowledge needs to be managed explicitly and updated continuously, since it is subject to constant change. Second, to identify security flaws in a specific system when the context knowledge is updated, its security requirements need to be accounted for. To support the automated validation of these requirements, they need to be maintained in a machine-readable form. Third, whenever violations of security requirements are detected, an immediate reaction becomes necessary: parts of the systems affected by vulnerabilities need to be identified; a suitable countermeasure needs to be determined and suggested to the security expert.

To address these challenges, in this paper, we present the SecVolution approach. The overall aim of SecVolution is to sustain the security of long-living systems whenever environmental changes have an impact on security properties. Our approach comprises three main components.

- *Security Context Knowledge* is expressed in terms of a *layered ontology* that allows the evolution of Security Context Knowledge to

* Corresponding author.
Email addresses: buerger@uni-koblenz.de (J. Bürger); strueber@uni-koblenz.de (D. Strüber); stefan.gaertner@adesso.de (S. Gärtner); ruhroth@uni-koblenz.de (T. Ruhroth); kurt.schneider@inf.uni-hannover.de (K. Schneider)
URL: http://jan.jurjens.de

[1] In this paper, we consider privacy as a security aspect, acknowledging the ontological debate around these terms.

https://doi.org/10.1016/j.jss.2018.02.003
0164-1212/

be managed and expressed in a formal manner. A central component of the layered ontology is an *upper ontology* of security-relevant concepts and their relationships.
- To detect threats in the system automatically whenever the Security Context Knowledge evolves, we enable the specification of *Essential Security Requirements* (ESRs). ESRs are amenable to an automated analysis against the Security Context Knowledge. To establish traceability between identified threats and the knowledge changes that provoked them, the analysis takes the *difference deltas* of the Security Context Knowledge as an input.
- Semi-automated reactions to co-evolve the security knowledge and the respective system models are specified using *Security Maintenance Rules*. As part of a Security Maintenance Rule, we employ model transformation rules to specify the system model evolution. A major benefit of this approach is that it can identify parts of the system models affected by security flaws automatically.

With this paper, we continue our ongoing work on the SecVolution approach (Ruhroth et al., 2014; Bürger et al., 2015), extending our earlier works in three main directions. First, the upper ontology presented here extends the one from our earlier work (Ruhroth et al., 2014) considerably by incorporating the results of a systematic literature review, allowing a more exact specification of security properties. Second, in our earlier work (Bürger et al., 2015), Essential Security Requirements were specified in natural language. With our new formalized notion of Essential Security Requirements, we provide a missing concept to enable an automated analysis of security requirements. Third, the aforementioned analysis, based on the formalized Essential Security Requirements and the Security Context Knowledge deltas, allows to detect potential threats in the system and map them to specific system artifacts.

To evaluate the extended SecVolution approach, we present a case study involving the open-source system iTrust. iTrust is a medical information system that fits well into our security setting.

As the evolution context, we use the privacy legislation of the European Union and Germany, comprising a set of privacy acts that have been changed repeatedly in the past years. Therefore, this setting is adequate to show the power of our approach in a realistic evolution scenario. The changes in the privacy legislation triggered changes in the underlying knowledge structure. The knowledge changes in turn called for changes to the system model of iTrust for recovering compliance to the privacy legislation. In sum, the evaluation shows the feasibility of our approach to identify security issues reliably and generate semi-automated reactions to security flaws.

The remainder of this paper is structured as follows: In Section 2, we sketch the SecVolution approach and define the scope of the research presented in this paper as well as relevant research questions. The evolution of environmental knowledge and a heuristic method to determine its impact on natural-language requirements is explained in Section 3.1. Based on this impact, the adaptation (or co-evolution) of the system model to retain a desired level of security is explained in Section 3.2 and Section 3.3. To evaluate our approach, we conducted a qualitative case study in Section 4 and discuss our results and insights. For this purpose, we used the medical care application *iTrust*. Related research in the field of security requirements and knowledge evolution as well as model co-evolution is listed in Section 5. In Section 6, we conclude our work and outline future research.

2 SecVolution approach and research challenges.SecVolution Approach and Research Challenges

According to Lehman and Ramil (2003), software evolution is the ongoing *progressive* change of software artifacts in one or more of their attributes over time. *Progressive* in this context means that the change results in improvement of the corresponding software. Each change preserves most properties (e.g. functionality and security) of the former system and is justified by a rationale. But changes may also lead to the emergence of new properties. Thus, evolution is caused by a wide variety of environmental changes such as technological changes, new stakeholders' needs, modified requirements and assumptions, changes in laws, rules as well as regulations, corrections of discovered problems and many others. Maintaining security of information systems by taking into account a continuously changing environment is therefore a challenging task in software engineering.

The term *co-evolution* is used in software engineering to describe the change of artifacts in response to a change in another artifact (cf. Mitleton-Kelly and Papaefthimiou, 2002). If artifact *A* evolves in response to changes in artifact *B, B* is called the causal artifact and *A* the effect artifact. Thus, co-evolution is the result of cause-effect changes between software artifacts. One reason for co-evolution is based on the fact that software artifacts depend on each other.

2.1. Overview of the SecVolution approach

The SecVolution approach is a holistic framework to deal with evolving knowledge in the environment of a software project. The overall goal is to restore security levels of an information system when changes in the environment put security at risk.

The SecVolution approach is the result of continuing research and extending the *SecReq* approach developed in our previous work (Houmb et al., 2009; Schneider et al., 2011; Jürjens and Schneider, 2014). As a core feature, SecReq supports reusing security engineering experience gained during the development of security-critical software and feeding it back into the model-based development process. To this end, SecReq combines three distinctive techniques to support security requirements elicitation as well as modeling and analysis of the corresponding system model: (1) Common Criteria (International Standardization Organization, 2007) and its underlying security requirements elicitation and refinement process, (2) the HeRA tool (Knauss et al., 2009) with its security-related heuristic rules, and (3) the UMLsec tool set (Jürjens, 2005) for secure system modeling and security analysis. This bridges the gap between security best practices and the lack of security experience among developers. However, a significant limitation of SecReq is that it cannot cope with evolution of the required security knowledge and, thus, has to be regarded as a "one–shot" security approach.

In SecVolution, to overcome this limitation of SecReq, the system's environment is monitored to infer appropriate adaptation operations. Fig. 1 depicts an overview of the resulting approach in the focus of this publication. Inputs to the approach are specification documents like (security) requirements, use cases, and misuse cases. Laws and regulations provide knowledge about general security obligations. Vulnerability databases contain knowledge about security best practices and also frameworks / algorithms that are known to be vulnerable and appropriate mitigations.

Security relevant knowledge for the system under consideration is elicited and captured in an explicit representation called *Security Maintenance Model* (SMM) (Gärtner et al., 2014). Security requirements that are defined on a coarse grained, *essential*, scale, are used to define security requirements independent from their concrete technical realization. The security knowledge is based on a upper ontology for security notions we provide. Changes of the system's environment (evolutions) and the system itself are captured as changes of the security knowledge. Evolution of the security knowledge is captured as difference information which triggers execution of appropri-

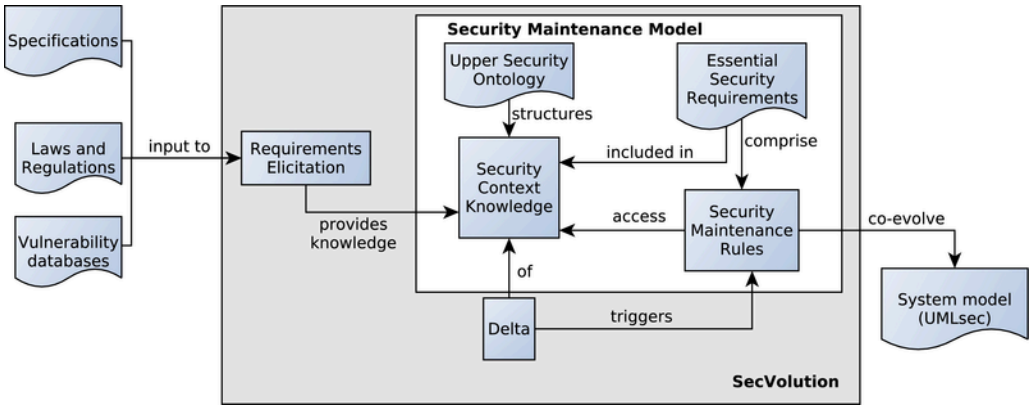

Fig. 1. Overview of the SecVolution approach.

ate co-evolution actions, called *Security Maintenance Rules* (SMR). Thus, application of SecVolution leads to a co-evolved system model that is compliant to the evolved environment again.

Monitored changes can trigger reactions, which leads to a co-evolution of security precautions and the corresponding system model. However, updating security precautions manually is time-consuming and error-prone. Therefore, there should be automated reactions to changes wherever possible. To this end, SecVolution uses security-enriched UML models, built upon the UML security extension UMLsec (Jürjens, 2005). To maintain an achieved security level, these models must be adapted to deal with environmental changes affecting security properties of the system. Current tool support to check security properties of UMLsec-enchanced models is available through our tool platform CARiSMA (Ahmadian et al., 2017).

SecVolution only requires the essential security knowledge and UML design models as input, therefore, it can be applied to all existing software systems for which these artifacts are made available. To support the elicitation of the security knowledge, an heuristic mechanism based on natural language processing of use-case descriptions (Gärtner et al., 2014) is available, which, however, is outside the scope of this paper.

2.1.1. Preliminary work

This publication extends our ongoing research on the SecVolution approach by contributing substantial new results. We first briefly summarize our preliminary work along the concepts introduced in our earlier discussion of Fig. 1.

An earlier, substantially smaller Upper Security Ontology of security-relevant methods, threats and mitigations was presented in Gärtner et al. (2014).

The ontology in this work was used to support a heuristic approach for the identification of security requirements in natural-language documents. Since the ontology was focused on a different purpose, it did not provide the required level of expressiveness for an automated co-evolution approach.

The plan to use layered ontologies for representing the Security Context Knowledge at different abstraction levels was presented in Ruhroth et al. (2014). This work presented a set of operations to describe ontology evolution as well as co-evolution strategies. However, it did not support an automated analysis, since the Essential Security Requirements included in the Security Context Knowledge were only given in natural language.

Co-evolving system models by using Security Maintenance Rules was first considered in Bürger et al. (2015). For illustration, this work used a particular rule-based technology, graph transformation. However, this work lacked a more formal representation of Security Maintenance Rules, including their relationship to changes of the Security Context Knowledge (*triggers* arrow in Fig. 1). Consequently, the condition under which co-evolution takes place was not explicitly defined, which lead to a substantial manual effort to identify the need for co-evolution.

In summary, our earlier work fell short to provide systematic means to support the semi-automated reaction to knowledge changes. In particular, it did not allow users (i) to specify Security Context Knowledge at the required level of expressiveness, due to the lack of a suitable upper ontology, (ii) to represent Essential Security Knowledge in a formalized way, and (iii) to automatically trigger the application of Security Maintenance Rules.

2.1.2. Contributions

This publication's focus, as reflected by the research questions we will present in Section 2.3, is to overcome the above mentioned limitations. We present substantial new contributions and use them to evaluate the applicability of the approach. Specifically, we make the following contributions:

- A new upper ontology that, based on a systematic literature review (Brereton et al., 2007), considerably improves the expressiveness of the Security Context Knowledge specification. Compared to Gärtner et al. (2014), the user can specify various new concepts and relations, such as *Security Properties* as well as containment relations between *Assets*.
- A newly introduced representation of Essential Security Requirements that aims to support the automated analysis of knowledge changes. This contribution is a substantial improvement to Ruhroth et al. (2014), in which these requirements were only available in natural-language form.
- A more systematic representation of Security Maintenance Rules that supports the automated reaction to knowledge evolution steps. Compared to Bürger et al. (2015), users can now specify triggers for Security Maintenance Rules based on changes of the Security Context Knowledge.

Based on these extensions, we also present an extended version of the case study originally presented in Ruhroth et al. (2014). The updated case study demonstrates the need for the new concepts and relationships of the upper ontology, and investigates how Essential Security Requirements and Security Maintenance Rules in their new form enable semi-automated co-evolution steps.

2.1.3. Fundamental concepts

The SecVolution approach is based on a number of fundamental concepts that interact with each other as shown in Fig. 1. In the following paragraphs, each concept is briefly explained.

Essential Security Requirements (ESR) capture the security needs of the system, focusing on potential attacks or threats (Braz et al., 2008), in a machine-readable form. ESRs are independent of concrete technologies and algorithms; their actual implementation depends on the given domain and environment. For example, the term "secure encryption" can require different algorithms and parameters when applied to an ordinary web application as when used in the banking sector. To implement a system that fulfills the given ESRs, extensive knowledge about current technologies, security principles, laws, and many others is needed.

Security Context Knowledge (SCK) is the security knowledge required to *instantiate* ESRs for a specific system. SCK includes, but is not limited to attacker types and their abilities, encryption protocols, and their robustness against various attacks. SCK is usually gathered from natural language documents of various kind, e.g. the security knowledge as part of the IT baseline protection guidelines proposed by the German Federal Office for Information Security (BSI, 2012), or attack and vulnerability reports as provided by the MITRE Corporation in the *Common Vulnerabilities and Exposures* (CVE) database (MITRE, 2013). Moreover, individual persons such as white hats or developers can also contribute to the SCK. SCK is supposed to evolve more often than the ESR, as knowledge about security techniques may change quickly and has direct impact on the security of the system.

Security Maintenance Rules. A change to the Security Context Knowledge reflects an evolution step of the considered system's environment. To cope with these changes properly, the development artifacts of the system need to be adapted accordingly by means of co-evolution. The connection between evolutions of the Security Context Knowledge and possible adaptations of the development artifacts is realized by *Security Maintenance Rules* (SMR). SMRs enforce preconditions that have to match with changes of the Security Context Knowledge and also with certain adaptations.

Security Maintenance Model. The core security information is aggregated in the *Security Maintenance Model* (SMM) which is connected to each activity of our approach as shown in Fig. 1. It mainly consists of *declarative* and *procedural* knowledge according to Robillard (1999). In SecVolution, the declarative knowledge describes what is known about a system and its environment with respect to security as well as about security problems and their solutions (semantic knowledge). It is concerned with concepts, objects, persons, events, facts, and their relationships. The Security Context Knowledge is part of the declarative knowledge and can be obtained from security as well as domain experts. In addition, the procedural knowledge comprises how to adapt a system or a set of system in order to fulfill the given security requirements. The Security Maintenance Rules are part of the procedural knowledge and can be derived from security guidelines and principles. Development artifacts such as requirements and system models are not part of the Security Maintenance Model.

2.2. Scope and research questions

In this paper, the overall research objective is to support co-evolution of the system based on environmental knowledge evolution. For this purpose, we focus on the artifacts, activities and data flows of the SecVolution approach as presented in Fig. 1.

Within the scope of this paper, we consider the evolution of the system environment. Changes to the environment may have an impact on security properties of a particular functionality implemented in the system. As a prerequisite, the knowledge about the environment needs to be made explicit in a form where it can be managed and continuously updated, which leads to the question ***(RQ1**:) How to explicitly represent the Security Context Knowledge (SCK), whose evolution may affect the security of the system?*

The resulting Essential Security Requirements (ESR) are used to determine elements of the system model which do not fulfill the security requirements anymore. This information is needed to decide which co-evolution of the model should be performed to retain associated security properties. For this purpose, predefined Security Maintenance Rules (SMR) must be selected properly. Thus, we need to know ***(RQ2**:) How to model Essential Security Requirements and Security Maintenance Rules?*

To perform the adaptation of the UMLsec system model semi-automatically, selected Security Maintenance Rules are parametrized using changes of the Security Context Knowledge as well as determined model elements. The determined co-evolutions guide developers on which parts of the design model to look at first and deepest. On this account, this leads to the question ***RQ3**: How to apply relevant Security Maintenance Rules for co-evolving the UMLsec system model?*

To examine our research questions, we focus on the major components of the SecVolution approach as illustrated in Fig. 2. In addition, we present a case study to show the feasibility of our approach.

3. Semiautomated co-evolution of security knowledge and system models

We now present the three main components of SecVolution in detail. First, to enable the systematic management of Security Context Knowledge, we present our layered ontology. Second, we consider the specification of Essential Security Requirements and their automated analysis based on the provided security knowledge and its changes. The analysis identifies security issues which may endanger a given security property. Third, to fix these properties, the system model is adapted, so that the system remains secure. If the adaptation of the system model cannot be performed automatically, software engineers are guided to change the system model semi-automatically.

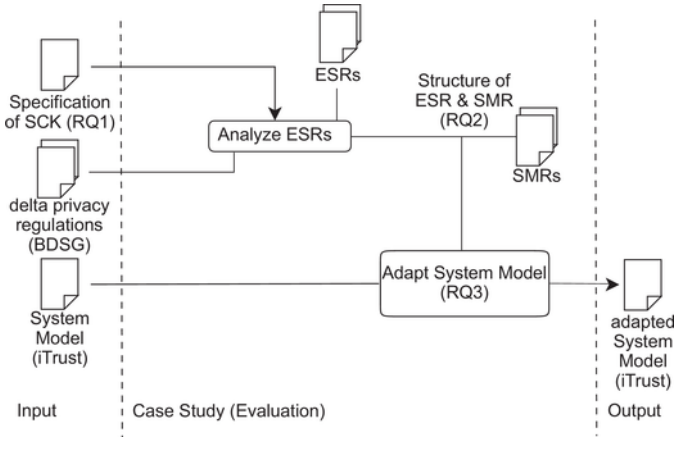

Fig. 2. Scope of our approach presented in this paper including research questions.

3.1. Capturing Security Context Knowledge (RQ1)

Security knowledge changes over time. Changes may lead to security loopholes as made assumptions and requirements are not longer valid and new attack vectors become possible. For example, the cipher suite RC4 has been popular over a long period of time and has been used in TLS for providing security for HTTP sessions. There have been a number of attacks shown on it. After the publication of an attack that can be carried out in merely 75 h (Vanhoef and Piessens, 2015), the use of RC4 has been prohibited in a RFC by the Internet Engineering Task Force (RFC, 2015). At that time, the estimation of TLS traffic relying on RC4 was 30%. HTTP is also used to implement APIs for the interoperability of distributed systems. Thus, systems using RC4 are principally vulnerable and need to be evolved. Based on incident reports or newly discovered knowledge such as new attacks, existing requirements must be adapted or new requirements must be elicited to prevent the threat. To properly react on knowledge changes, knowledge must be modeled explicitly to make it capable for our assessment approach.

The Security Context Knowledge (SCK) needed in our approach is modeled in an ontology using security-relevant concepts and their relationships. The motivation to use ontologies for this purpose was twofold: Regarding security knowledge, we have to deal with the *unknown unknowns* (McManus and Hastings, 2005). Therefore, we decided to use ontologies for knowledge modeling, because they are based on the open-world assumption. Second, reuse of knowledge between ontologies of is natively supported by an import mechanism, so that the layering of ontologies with an arbitrary number of layers is directly supported (Ruhroth et al., 2014).

Systematic literature review

To define an upper ontology, we conducted a systematic literature review (SLR). SLR is an empirical method used to aggregate, summarize, and critically assess all available knowledge on a specific topic (Kitchenham and Charters, 2007). In our case, we searched for scientific publications related to the modeling of security-specific knowledge in the context security management or software and systems modeling. In particular, we addressed publications in which concrete ontologies for the modeling of knowledge related to IT security are described.

To find the relevant publications, we used the literature databases *ACM Digital Library, IEEE Xplore, ScienceDirect*, and *SpringerLink*, since they cover the largest part of scientific journals, conferences, and workshops in the domains of software engineering and knowledge management (Brereton et al., 2007). The search query for the automated search was obtained by combining the search terms *Security* (in title), *Information System, Software* (in title or abstract), and *Ontology, Metamodel* (in title, abstract or text). This search yielded a total population of 284 publications, which we subsequently filtered to retain only those that (i) were available in English, (ii) describe an existing, practically applicable approach (rather than e.g. a position statement), (iii) address the modeling, application, or acquisition of security knowledge in software engineering, and (iv) were not specific to a particular domain. The remaining 46 publications contained 26 security-related ontologies, that we analyzed in detail to aggregate the minimal set of concepts required to enable the modeling of security knowledge.

Ontology analysis

In the analysis of the 26 ontologies, our objective was to answer the following research question: *What is the minimum set of terms required for modeling security knowledge to enable a heuristic security analysis of development artifacts?* By focusing on the minimum set of terms, we address a key prerequisite for the design of our ontology, related to *ontological commitment*.

Ontological commitment (Gruber, 1995) is an important principle in ontology engineering. To support flexible use of a created ontology, it proposes to impose as few restrictions as possible. A minimal ontological commitment is achieved if the ontology contains only the essential and most general terms in the considered domain. This allows others to use the ontology in their particular applications, specializing it where necessary. Based on our SLR, we achieved a minimal ontological commitment by extracting the essential and most general terms. A term fulfills this criterion if it occurs in multiple ontologies, or if it was pointed out as particularly relevant in the underlying ontology's description.

To address the research question, we developed a classification of all terms included in all ontologies. To this end, the terms were assigned to classes, based on the descriptions in the publications from which they were obtained. Various concepts occur under different terms across publications, mostly due to the different research backgrounds of the authors. For instance, in most considered ontologies, the term *asset* is used to refer to a protection-worthy object; yet, in some cases, the terms *affected element, resource*, or *information* were used. Our classification represent all of these terms as one particular class *asset*.

Result upper ontology

From the classification, we obtained the security-relevant classes for our upper ontology. Fig. 3 shows the resulting upper ontology in VOWL 2 syntax; circles and lines denote classes and relationships between them. A double border represents equivalent classes.

The classes *system, access point, asset*, and *trust level* represent system-specific knowledge about the system and its parts. These aspects are addressed by the 10 of the 29 considered ontologies, in which the terminology is subject to some variability. For instance (Elahi et al., 2009) distinguish the system parts *product, component*, and *function*. According to Swiderski and Snyder (2004), an access point is an integral part of a system, application, module, or component.

The class *security property* describes security requirements to an asset of the considered system. 15 out of the 29 considered ontologies specify a corresponding term. The relationship between a security property and an asset is highlighted in several ontologies, such as the one by Dubois et al. (2010).

The classes *threat* and *attack* are dedicated to attack and threat modeling. Corresponding terms are included in 26 out of the 29 considered ontologies; 21 of them also represent vulnerabilities or weaknesses of the considered system explicitly. Usually, a vulnerability is related to a particular attack or threat. Moreover, the proposed ontology assigns an intent to attackers, which is required to specify the goal or motivation of attackers. Comparable terms are included in the ontologies of, for instance, Elahi et al. (2009) and Miede et al. (2010).

The class *action* is used to specify the concrete steps performed by an attacker to exploit a specified security weakness. Modeling these concrete steps is necessary to facilitate the detection of weaknesses where the order of steps performed by an attacker matters (Jhawar et al., 2015). 9 out of 29 ontologies considered this aspect; the most detailed description is found in the one by Elahi et al. (2009), in which an attack consists of a sequence of steps, called *malicious actions*.

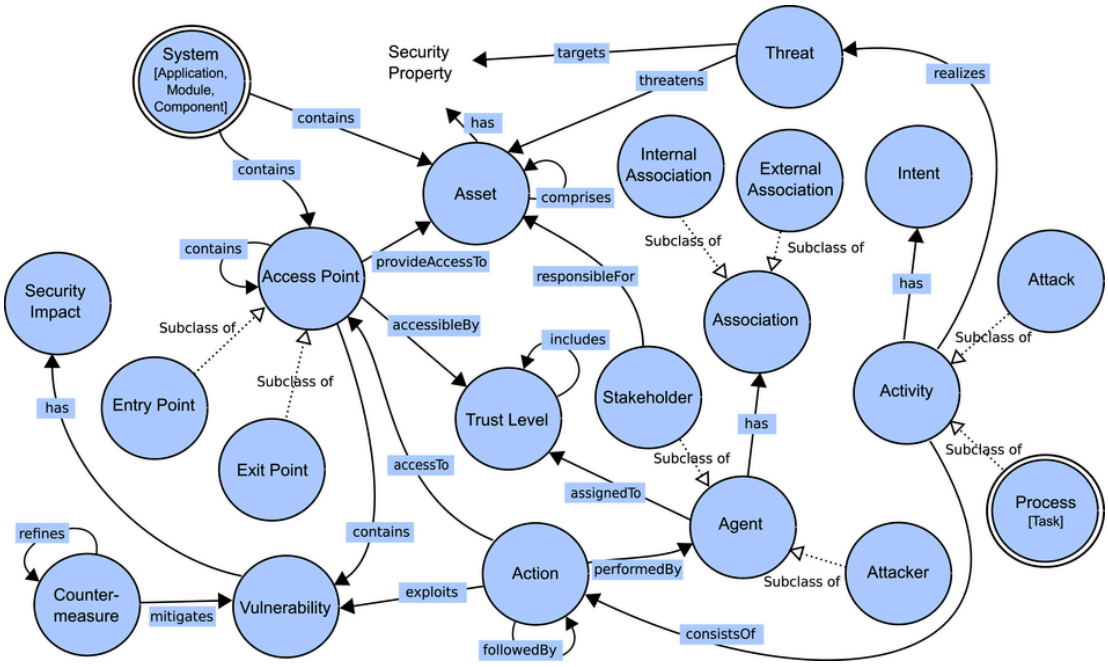

Fig. 3. Upper ontology of security concepts and their relationships.

The classes *stakeholder* and *process* are dedicated to users and business processes. 9 out of the 29 ontologies consider this aspect. *Stakeholder* is a sub-class of *agent* used in particular for representing the users of the considered system. Based on the ontology of Baras et al. (2014), stakeholders can be assigned responsibility over *assets*. In addition, agents can refer to *components* to perform specific actions. *Process*es represent a type of activity with consists of a number of actions. Related terms are found in the ontologies of Launders and Polovina (Launders and Polovina, 2013; Mouratidis et al., 2003).

By defining fundamental security concepts and relations, this upper ontology is the foundation of a layered ontology that we use to represent security knowledge in general and in a specific system. The lower layers are more system specific: In the example shown in Fig. 4, it is represented that a specific communication path uses RC4 as an encryption algorithm to achieve secure encryption. Thus, a part

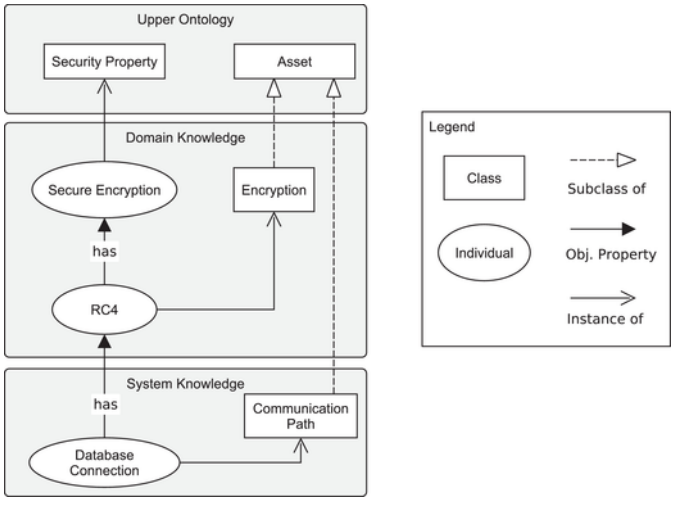

Fig. 4. Example of Security Context Knowledge to provide an encryption algorithm.

of the security knowledge is specific for the system at hand. Adapting the knowledge to other systems is possible with regard to system- or domain-independent parts of the security knowledge.

3.2. Determine impact of knowledge evolution to the system (RQ2)

Using the specification of Security Context Knowledge as illustrated in the previous section, evolved requirements or new attacks can be used as a trigger to assess the impact on the system model under consideration: After an evolution step, it needs to be checked if the system model needs to be co-evolved.

In Fig. 5, we outline how evolution and co-evolution are related to each other in the SecVolution approach. We assume that the initial system model (SyM) fulfilled all security requirements regarding the *Security Context Knowledge (SCK)*, as was ensured by an upfront security analysis.

After an evolution of the Security Context Knowledge has taken place (ev_{SCK}), the challenge is to assess the related knowledge changes and, if necessary, adapt the system model (ev_{SyM}) such that the adapted model (*SyM'*) meets all *Essential Security Requirements* (ESRs) regarding the updated security knowledge (*SCK'*). To this end, our approach can discover appropriate co-evolutions that, when applied to the system, can recover the system's security.

This is done semi-automatically by leveraging information about the difference in the knowledge in order to infer necessary adaptation steps.

Fig. 6 gives an overview of the process how an emerged knowledge difference is analyzed and possible co-evolution actions are inferred. The input to the algorithm is a knowledge difference, emerging by the evolution of the Security Context Knowledge. The knowledge difference is analyzed to discover a change event which is relevant for further investigation. For example, this can be done using model differencing methods (Kehrer et al., 2012). Possibly affected Essential Security Requirements are identified. Evolved Security Context Knowledge parts can then be used by Security Maintenance Rules to co-evolve the system.

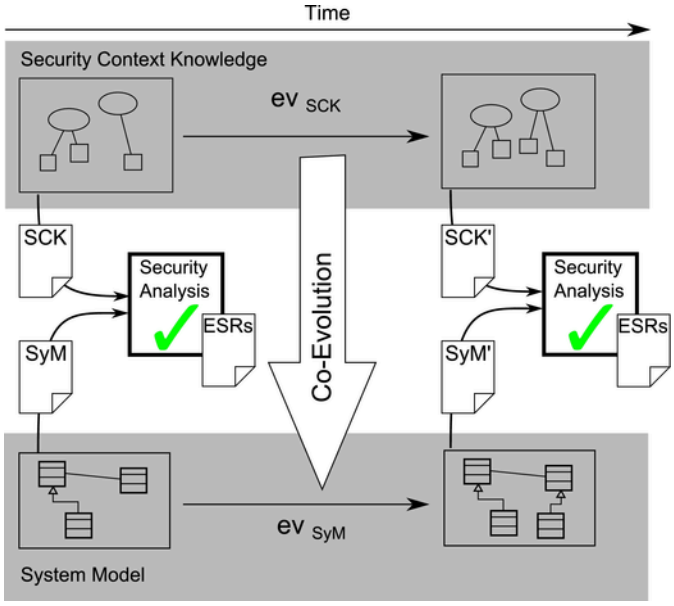

Fig. 5. Relationship between evolution and co-evolution as used in the SecVolution approach.

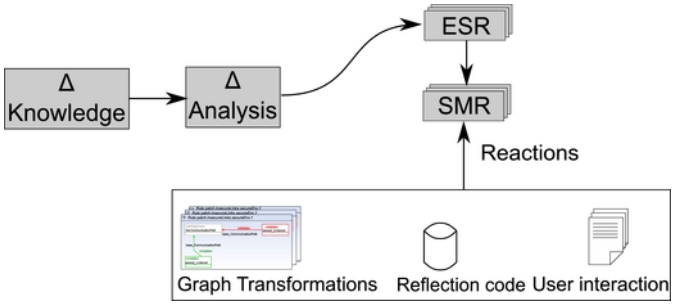

Fig. 6. Concepts used in the co-evolution approach and their relation.

Co-Evolutions can be realized by various methodologies. These and all relevant components are presented in detail during the next paragraphs.

Essential Security Requirements

An *Essential Security Requirement* (ESR) consists of the following components:

- its name
- A detailed description concerning vulnerabilities and mitigations
- A formal selection criterion to check if the current Security Context Knowledge exhibits this vulnerability

Whenever the Security Context Knowledge evolves, this is considered an *event* that needs to be analyzed. The selection criteria of the available Essential Security Requirements can be evaluated to determine the affected ones. Further investigation and eventual application of co-evolution steps are handled by Security Maintenance Rules.

Security Maintenance Rules

Security Maintenance Rules (SMRs) build the connecting link between changes to the Security Context Knowledge (evolution) and necessary adaptations to the system model to recover its security (i.e. compliance to Essential Security Requirements, co-evolution) (Bürger et al., 2014). Given a change to the environmental knowledge of the regarded system, a sequence of co-evolution operations on the system model needs to be inferred from the evolution operations on the security knowledge.

To support this, Security Maintenance Rules are structured following the *Event-Condition-Action* principle (Dayal, 1994), which means that a rule follows this schema:

ON Event IF Condition DO Action

Event indicates the event given by a change of the Security Context Knowledge (SCK) which can be used to get relevant Essential Security Requirements (ESR) and UMLsec stereotypes. Using the *Event*s, possible relevant Security Maintenance Rules can be selected, but deeper investigation is needed, which is defined by the second part:

The *Condition* examines if the system is in a non-compliant state. This is realized using model queries, compliance checks, etc. with respect to the (unchanged) ESRs. Preconditions ensure that queries and possible later actions are only applied to a system model that principally is annotated with the respective security property. A query can make use of the security knowledge difference or impact trigger data. For example, Security Maintenance Rules addressing state charts should only be applied to a model containing appropriate information.

The *Action* part defines steps that can be applied to recover Essential Security Requirements compliance. If a query has discovered one ore more model flaws, appropriate *reactions* can be triggered. A reaction consists of a sequence of steps. As depicted in Fig. 6, a reaction step can be either a direct manipulation of the system model by using graph transformation, Java code (EMF manipulation) or user interaction with the security expert, either to realize changes manually or further parameterize reactions (like defining new class names, choose desired position of new elements in model hierarchy, etc.). All components can have input- and output data and thus share parameters and/or results to optimize/parameterize subsequent steps.

Every Security Maintenance Rule focuses on the preservation of one or more Essential Security Requirements (ESR). To realize this, every Security Maintenance Rule makes use of *model queries* to identify model elements violating corresponding ESRs. It further infers *reactions* to correct the violations which actually is done using *graph transformations, user interactions* and *reflection code*. Moreover, graph transformations can be extended by using Java code as discussed in our previous work (Bürger et al., 2015).

In the following, we introduce main concepts and techniques as used by our approach.

Model Queries

To retrieve specific information from a model, model queries can be used. Since system models can be interpreted as graphs (Bürger et al., 2015), various techniques based on graph algorithms can be incorporated to investigate properties of a given model. Model queries are a widely used concept (Habela et al., 2008; Ujhelyi et al., 2015). A model query is carried out by firstly providing a query string based on a query language and a model as input to an evaluation algorithm. Execution of this algorithm determines a set of model elements matching the query, which also can be empty. To check if a model violates certain properties, one can proceed as follows: the non-compliance to this properties is modeled as query. If the query result is empty, it means that the model does not contain non-compliant elements and thus is compliant to the property in question. By now, our approach focuses on the following possibilities to query models. First, we can formulate queries by a graph transformation and its underlying matching algorithm (Bürger et al., 2015). Second,

we can make use of our tool platform for risk and compliance checks, CARiSMA (Ahmadian et al., 2017)) which supports analyzing models e.g. in UML using approaches such as OCL to formalize the security properties under investigation.

Graph Transformation

Graph transformation is a well established concept to describe change operations of graphs in a formal way, mainly used for describing changes, synchronizing two graphs, e.g. coming from different meta-models or generating code out of a software system model. Graph transformations can also be used to conduct model queries by designing graph transformation rules where the left-hand side (LHS) equals the right-hand side (RHS) (Bürger et al., 2015). To model and execute graph transformations, we make use of the graph transformation framework Henshin (Arendt et al., 2010). Henshin has a unified view on the LHS and RHS of a transformation rule. In a nutshell, mappings between LHS and RHS are represented as so-called *actions*. These actions are annotated as stereotypes in a graphical representation. We shortly introduce the fundamental action types. ⟨⟨`preserve`⟩⟩ means that an element is member of the LHS as well as the RHS. ⟨⟨`create`⟩⟩ means that an element is only member of the RHS. ⟨⟨`delete`⟩⟩ means that an element is only member of the LHS. Henshin features a powerful API that provides access to results of a match and thus gives a flexible way to interpose Java code between rule executions. For example, *partial match* allows us to use transformation rules with fewer nodes. Here, usage of partial match as well as the Henshin API allows us to design more abstract transformation rules that are parameterized using program code with information gathered from model queries and the knowledge delta. We make use of Henshin in two ways. First, we use Henshin to carry out model queries. This is accomplished by modeling graph transformation rules solely using ⟨⟨`preserve`⟩⟩ type nodes. This way, we achieve that there is no real modification of the underlying model carried out, but the matcher as part of Henshin is used as a pattern matcher for the given model.

In Fig. 7 we present a model query which is used to search a state in a state chart by a given name.

Thus, every match of such a *query* rule means that the respective pattern has been found in the model.

Second, we also use Henshin to carry out co-evolutions.

Additional details on Henshin and how we use it to find model flaws is in our previous work (Bürger et al., 2015; 2015).

Java access to EMF meta model implementation of UML

The Eclipse EMF implementation (Eclipse Foundation, 0000) of meta models offers a rich set of methods to get and set properties of model elements and also traverse models. For example, regarding state machines, for a given state, the method `getOutgoings()` returns the set of transitions that have the given state as source. Using methods like this and graph traversing algorithms, customized exploration of paths through the state machine is feasible. Moreover, using methods provided via the meta model such as `add()`, `remove()` and auxiliary methods of Ecore, the model can directly be manipulated.

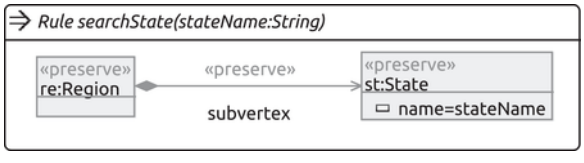

Fig. 7. Henshin model query to search a state in a UML state chart by its name.

3.3. Semiautomatic co-evolution of models (RQ3)

After a knowledge evolution has emerged and the evolution to react to has been identified as discussed in the preceeding section, the system under consideration needs to be co-evolved. In this section we briefly introduce the proposed algorithm for co-evolution. The algorithm was first introduced in previous work (Bürger et al., 2015) and is presented in Fig. 8. Differing from the usual flow chart notation, dashed lines represent important data flows. The goal of the algorithm is to determine violated Essential Security Requirements (ESR) and propose necessary recovering steps. A core assumption of the SecVolution approach is that the system was secure and compliant to all Essential Security Requirements with the respective Security Context Knowledge (SCK) when it was initially designed. To validate this, numerous approaches exist, e.g. the UMLsec approach and tool support (Jürjens, 2005; Ahmadian et al., 2017).

In the first place, the algorithm is triggered by a Security Context Knowledge evolution (i.e. a ΔSCK exists (Ruhroth et al., 2014)) or an impact analysis that derived a potential flaw of the information system (event part of the Security Maintenance Rule). A list of candidate Essential Security Requirements is determined. This can either be done as part of the event because a preceding impact trigger has determined it or as part of model queries in the condition part. In the first step of the algorithm, model queries are used to investigate if there are Essential Security Requirements endangered (condition part of the Security Maintenance Rule).

This is realized by evaluating model queries, supported by the trigger information as well as the existing Essential Security Requirements. The model queries here are constructed in a way that they determine model elements that violate a given Essential Security Requirement (ESR).

The set of violating model elements is used by the succeeding activity. Here, the *reactions* part of the Security Maintenance Rules is brought into action. As there can be multiple reactions to treat a given maintenance object or query, every reaction is checked with respect to its applicability. Every reaction step can have input and output parameters. Thus, reactions can be composed by (re-)using modularized reaction steps.

To actually realize co-evolutions, a reaction of a Security Maintenance Rule has to be *applied*. This means that the reactions are carried out at the system model. As discussed above, there are different kinds of reactions that can be used as a co-evolution. This can be done by executing graph transformations to alter the model directly. Using the *partial match* mechanism of Henshin, it is possible to preallocate nodes of the transformation rules with concrete model elements. Thus, we just need a reduced set of simple rules which we can concrete with information gathered in preceding steps.

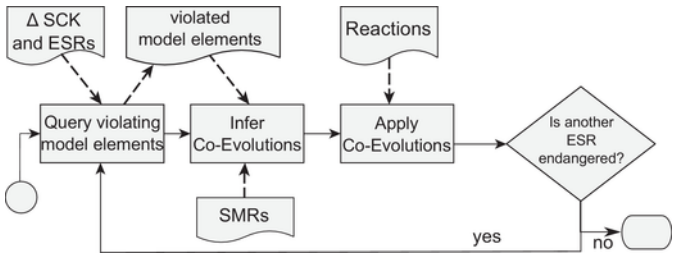

Fig. 8. Overview of our algorithm to co-evolve system models.

As shown in Fig. 6, a reaction step can also include user interaction. One kind of interaction is to request input from the security expert, such as a name for a model element that needs to be created. Another kind of interaction is to *instruct* the security expert to do something to recover the system's security that cannot be applied directly by changes to the model, for example to instruct all users to pick new passwords after a new policy is introduced. To enable complex reactions that cannot easily expressed using graph transformation (i.e. path expressions, clone operations with containment hierarchy), reaction steps make use of EMF meta model and utility methods. If there is only one reaction applicable for a given Security Maintenance Rule, it can be applied semi-automatically. Otherwise, a security expert is presented the alternatives and needs to choose the Security Maintenance Rule that is suitable in the given context. In summary, the following situations can occur: (1) The reaction is determined fully automatically, (2) the reaction requires some user input or (3) the user needs to manually adapt the system. If there is no further ESR that is potentially violated after applying the co-evolutions, the algorithm terminates. Otherwise, the algorithm starts over with the activity *Query violating model elements*. Thus, the algorithm is repeated as long as there is an ESR that is potentially violated.

Afterwards, it has to be checked if still all Essential Security Requiements (ESR) are fulfilled. If an ESR is now violated that was respected before execution of the algorithm, or if there is an ESR that still is violated, the assistance of the security expert becomes necessary. The security expert needs to investigate the ESRs, eventually manually adapt the model to prevent side-effects, and re-run the algorithm. In particular, this becomes necessary if two security requirements are to be applied that (logically) contradict each other (such as non-repudiation and anonymity).

3.4. Implementation

Our approach is supported by an implementation prototype, which uses a number of underlying frameworks and tools. As common base, the prototype as well as its supporting approaches use the Eclipse Modeling Framework EMF (Eclipse Foundation, 0000).

The upper ontology as well as ontologies of domain levels are modeled using Protégé (Stanford Center for Biomedical Informatics Research (BMIR), 0000). Protégé makes use of OWL-API to persist ontologies, e.g. in OWL-XML format. To bridge the gap to EMF, we implemented a transformation which transforms ontologies instantiated from the OWL-API meta model to the Ecore meta model provided by W3C[2]. To determine changes in the knowledge, we access the ontologies using EMF tooling. We use the SiLift tool for semantic differencing (Kehrer et al., 2012). The user provides high-level differences which should be detected in terms of a graph transformation language. SiLift calculates the difference between two ontologies and can be controlled using an API.

System model co-evolution steps are realized using the Henshin graph transformation language (Arendt et al., 2010). This is realized by modeling rules that alter UML models which can be further customized, which leads to a high flexibility while the number of rules needed can be kept low. In addition, we use Henshin to specify model queries. Additionally, in a few instances, we extended them with supplementary Java code to support queries on *paths* between two model elements.

Based on the implementation, we conducted the case study presented in Section 4.

[2] https://www.w3.org/2007/OWL/wiki/MOF-Based_Metamodel (accessed Dec 20, 2017).

4. Case study

To evaluate our approach, we conducted a case study in healthcare, a particularly security-critical domain, since patient records are subject to strict confidentiality requirements. We used three sources of security knowledge: First, *Common Weakness Enumeration* (CWE, (MITRE, 2017)), a dataset of common software security weaknesses. The knowledge represented in CWE is provided by about 50 different organizations and companies of the software engineering community.

Second, the German privacy laws, which have been subject to various changes since 1990. By declaring specific types of data as particularly protection-worthy, these laws present a suitable source of security knowledge evolution.

Third, knowledge about a particular encryption algorithm, which was rendered ineffective by newly discovered attacks. To study the ability of our approach to support semi-automated co-evolution for adapting the system after knowledge changes, our discussion focuses on the three research questions outlined in Section 1.

4.1. Case study design

Subject system

We studied the open source healthcare system *iTrust*. iTrust is a role-based medical information system implemented as a web-based application. Its purpose is to provide patients with medical information, let medical staff organize their daily work, and provide a messaging system so that all users can communicate with each other. The iTrust project was initiated at North Carolina State University and is currently maintained by the Realsearch Research Group (Meneely et al., 2012). It is a fully operational system, whose development artifacts, in particular, requirements and code, are publicly available. Its 51 use cases are documented in natural language (we used version 23 of the requirements and the corresponding code version 17 for our study), some of them addressing privacy and security issues regarding personal and medical data.

As a large system maintained over a long period of time, iTrust is a well-suited example of a long-living system with substantial practical relevance: First, it underwent organizational changes. Being in use since 2004, responsible architects and developers changed several times, and functional requirements have also changed.

Second, it was subject to substantial functionality changes. Over the years, new use cases have been implemented and others have been removed, according to continuously changing requirements. For example, in requirements version 27, the system is implemented based on 39 use cases, while the overall sum of documented use cases is 79 (cfer. Realsearch Research Group North Carolina State University (2017)).

Third, it is affected by the evolution of well-documented environment changes, in particular, the law changes considered in this section. As a healthcare system, iTrust is particularly prone to privacy and security issues (Bowman, 2013). All of these properties are also typical for a long-living system used in industry.

Since the iTrust project does not provide any models, the design models were obtained in a reverse engineering process. The process was conducted manually, as automated reverse engineering approaches typically produce implementation-centric models, being too low-level for our purposes. To avoid the risk of a biased team that (unknowingly) optimizes the model towards the approach or vice versa, reverse engineering was performed by an independent team. The team consisted of a researcher with 10 years of industry experience as software architect, a PhD student and two student assistants,

all of them being computer scientists. The overall team size of 4 people is a typical team size in practical settings (Schwaber and Sutherland, 2017). The reverse engineering team did not take part in developing the prototype as well as in implementation of the approach.

The re-engineering was started out by inspecting the code base. While doing so, a three-tier architecture was identified and thus taken into account, so that the design model reflects this architecture. The overall system model included class diagrams, deployment diagrams, and state charts, and which are common diagram types to specify structure and behavior, respectively (Hutchinson et al., 2014). The class diagram is based on the architecturally relevant code parts. By mapping relevant parts of the use case descriptions to operations, supposed call flows were identified. The correctness of the state machine was checked by running test cases on an iTrust instance.

The team defined a test plan based on use cases found in the iTrust documentation. The tests were carried out on an iTrust instance, by manually checking if the sequence of states an control flow corresponds to the model.

Security knowledge

Our approach supports semi-automated co-evolution for reacting to changes in the security knowledge. The security knowledge considered in our case study comes from three sources:

First, we selected five CWE entries, three of them being part of the 2011 CWE/SANS Top 25 most dangerous software errors[3]. These CWE entries are representative examples of weaknesses that can be addressed at the system design level.

In our case study, we modeled Security Maintenance Rules (SMRs) to detect and react to presence of these weaknesses. Specifically, we considered the following CWE entries:

- CWE-284: Improper Access Control
- CWE-306: Missing Authentication for Critical Function
- CWE-311: Missing Encryption of Sensitive Data
- CWE-326: Inadequate Encryption Strength
- CWE-732: Incorrect Permission Assignment for Critical Resource

Second, as source for knowledge evolution, we used the German Federal Data Protection Act (in German: Bundesdatenschutzgesetz, BDSG (Bundesministerium des Inneren, 2005)). In particular, we considered the history of recent changes to this law, and their security impact to the system model of the iTrust system. The security impact is assessed against the five selected CWE entries explained above.

Third, as further source for knowledge evolution, we consider the common scenario of an encryption algorithm that is discovered to be insecure.

In the following, we present the knowledge evolution in detail and show how assessment of it is carried out.

Knowledge evolution

We considered the effect of the introduction of a common privacy understanding in the European Union Directive 95/46/EC. This directive required adjustments of the national privacy laws, including the BDSG. Starting from 1995, the BDSG had to be altered several times to be fully compliant to the European Directive. In addition to the regular notion of *private data*, the 2001 version of the BDSG introduces *special categories of personal data*, including data about racial or ethnic origin, political opinions, religious or philosophical convictions, union membership, health and sex life (cf. Bundesamt für Datenschutz, 2014). The access to this kind of data needs to be more restrictive as enforced in section 13 par. 2 BDSG (translated):

> The collection of special types of personal data (Section 3 (9)) is permissible only in so far as [...] 7. Such collection is necessary for the purposes of preventive medicine, medical diagnosis, health care or the administration of health services and the processing of these data is carried out by medical personnel or other persons who are subject to an obligation to maintain secrecy [...].

Furthermore, we consider a knowledge evolution step related to the *RC4* RFC encryption algorithm, which was declared insecure after the discovery of severe attacks in 2015. More details are found in Section 3.1, where we first introduced RC4 as an example.

4.2. Assessing security impact using ontologies

The aforementioned changes have a significant impact to iTrust, since it processes different sorts of privacy-relevant data. As a first step in adapting the system to remain compliant, we assess the impact of context evolutions according to the CWE entries introduced above. To this end, we define for each CWE entry an Essential Security Requirement (ESR) and a corresponding Security Maintenance Rule (SMR), as summarized in Table 1.

Essential Security Requirements are given in terms of their name, description, and selection criterion; the selection criterion is at the same time the *ON* part of the corresponding Security Maintenance Rule.

In addition, a Security Maintenance Rule contains an *IF* and a *DO* part. For brevity, the *ON, IF*, and *DO* parts are shown in a pseudo-code notation, representing a more detailed implementation using Henshin-based graph transformation queries as well as Java code. In the implementation of the *ON* part, to obtain what are in fact difference deltas of two revisions of the same ontology, we use the semantic model differencing approach SiLift that produces these deltas. More details on these implementation aspects are found in Section 3.2.

Each Essential Security Requirement resembles a security property that is subject of an CWE entry. A CWE entry (MITRE, 2013) contains the following aspects, among others: (1) a description of the security issue, (2) consequences, (3) applicability to specific programming languages and development process phases, and (4) possible mitigations.

In our setting, the system becomes subject to a certain CWE entry after the environment has changed: either a change of the domain knowledge (e.g. an encryption algorithm becomes insecure) or a law change (introduction and handling of the additional notion for personal data). In what follows, we present the specification of these changes using excerpts of the Security Context Knowledge.

Fig. 9 illustrates the relevant ontology parts that enable the detection of a violation of ESR1, *Secure Encryption*. The system-specific ontology part is not shown explicitly, since no knowledge about the specific system is necessary in this case. The respective Security Maintenance Rule (SMR) is designed that way that the system model is queried for all relevant elements. Note that for the sake of clarity, we only present ontology and model elements that are relevant for the study's scope.

The knowledge change presented here is induced by the general announcement that the encryption algorithm RC4 should no longer be used (RFC). RC4 is actually modeled as an encryption individual able to provide the security property *Secure Encryption*. The knowledge evolution consists of adding a threat *Weak encryption* to it, as well as defining *Disclose data* and *Hack key* as the relevant attack

[3] http://cwe.mitre.org/top25/ (accessed Dec 20, 2017).

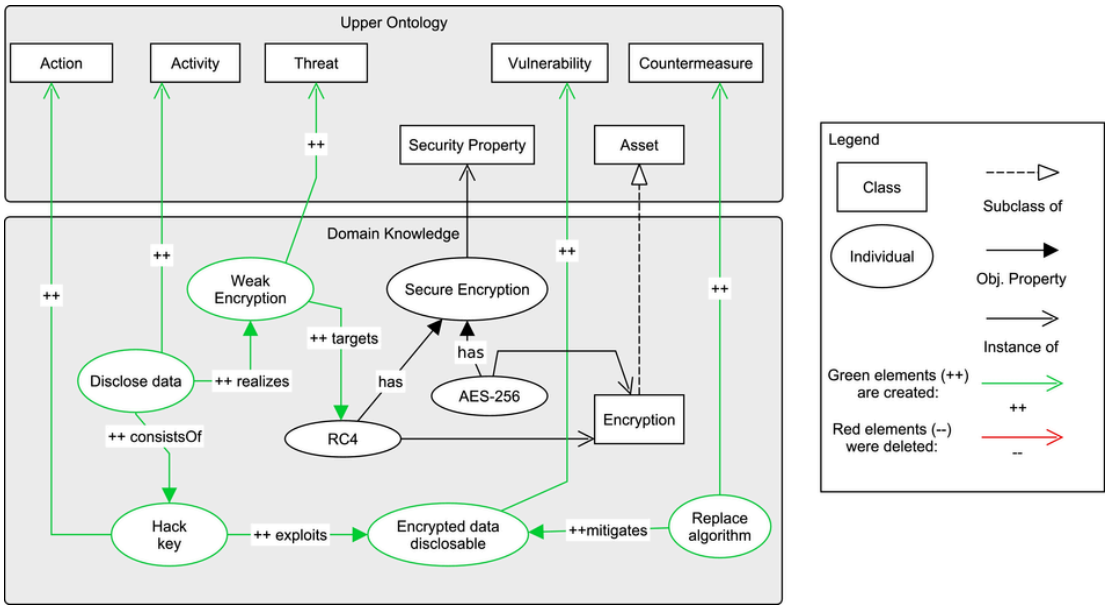

Fig. 9. SCK excerpt and changes regarding ESR1 (see Table 1).

Table 1
Essential Security Requirements and respective Security Maintenance Rules.

	ESR1	ESR2	ESR3	ESR4	ESR5
Name	Secure encryption	No access to unauthorized users	Authentication for critical functions in place	Encryption applied to resources	Locking is in place
Description	CWE-326	CWE-284	CWE-306	CWE-311	CWE-732
Selection Criterion SMR: ON	`ADD entity t:Threat WHERE t.domain= ”Secure Encryption”`	`DEL link l:x→y WHERE l = ”accessible By”`	`ADD link l:x→y WHERE y = ”Critical Function”`	`ADD link l:x→y WHERE y = ”Encrypted Persistence”`	`ADD link l:x→y WHERE y = ”Lockable Data”`
SMR: IF	`EXISTS p: communication Path WHERE p uses c:cypher WHERE t threatens c`	`!empty(forEach (s:State, traceAccess (s,x,y)))`	`!operation(x). stereotypes. CONTAINS(rabacRequire)`	`class(x). stereotypes. CONTAINS(encrypted Persistence)`	`class(x). attributes. CONTAINS(”locked”)`
SMR: DO	`FIND c’:cipher WHERE c’.threats .isEmpty; p.forEach(SET p.uses = c’)`	`s’:state= Parent(s). clone(); s.incoming. stereotypes. ADD(ensureRole) s’.remove(s);`	`r:right= UserInput(); operation(x). stereotypes. ADD(rabac Require(r))`	`class(x). stereotypes. ADD(encrypted Persistence)`	`class(x). attributes. ADD(”locked”)`

vector. Thus, the system potentially now exhibits the vulnerability *Encrypted data disclosable*. A mitigation *Replace algorithm* to this vulnerability is also introduced.

The knowledge evolution shown in Fig. 10 concerns ESR2, *No access to unauthorized users*. As environmental change, we consider the privacy regulation change as introduced in Section 4.1. The notion of *Special Personal Data* is newly introduced. Moreover, the law states that medical data is to be treated as special personal data.

iTrust stores that information inside a *Patient* asset which is accessed by the *Patient View* system component. iTrust comprises a hierarchical role model of which the relevant elements are depicted. The system handles different types of licensed health care personnel, such as designated licensed health care personnel (DLHCP) and licensed health care personnel (LHCP). An additional role is unlicensed authorized personnel (UAP). Initially, all roles have access to the patient view component. Consequently, Fig. 10 shows a *accessibleBy* connection from the asset to the enclosing role HCP (health care personnel).

To re-establish compliance, the Security Context Knowledge is updated so that *Patient* is now treated as *Health-related* data and UAP should not be allowed to have access to the patient view anymore. This is accomplished by a more fine-grained rights assignment: the coarse-grained right assigned to HCP is removed and fine-grained rights to DLHCP and LHCP are added.

Fig. 11 deals with three Essential Security Requirements: ESR3 (*Authentication for critical functions in place*), ESR4 (*Encryption applied to resources*), and ESR5 (*Locking is in place*).

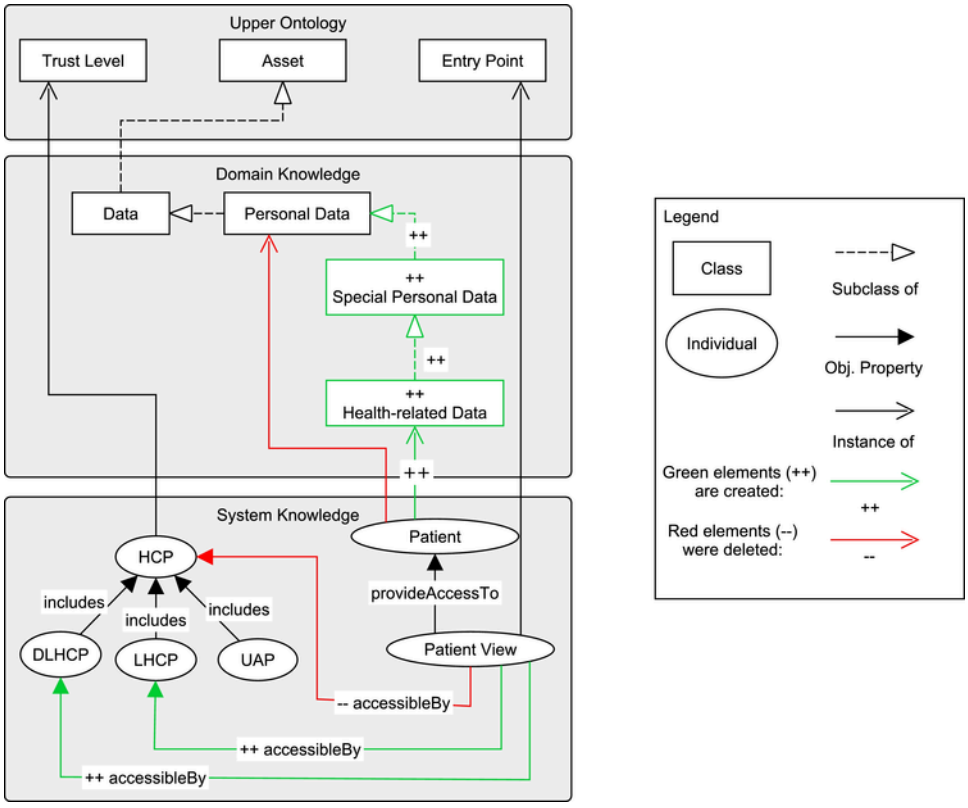

Fig. 10. SCK excerpt and changes regarding ESR2 (see Table 1).

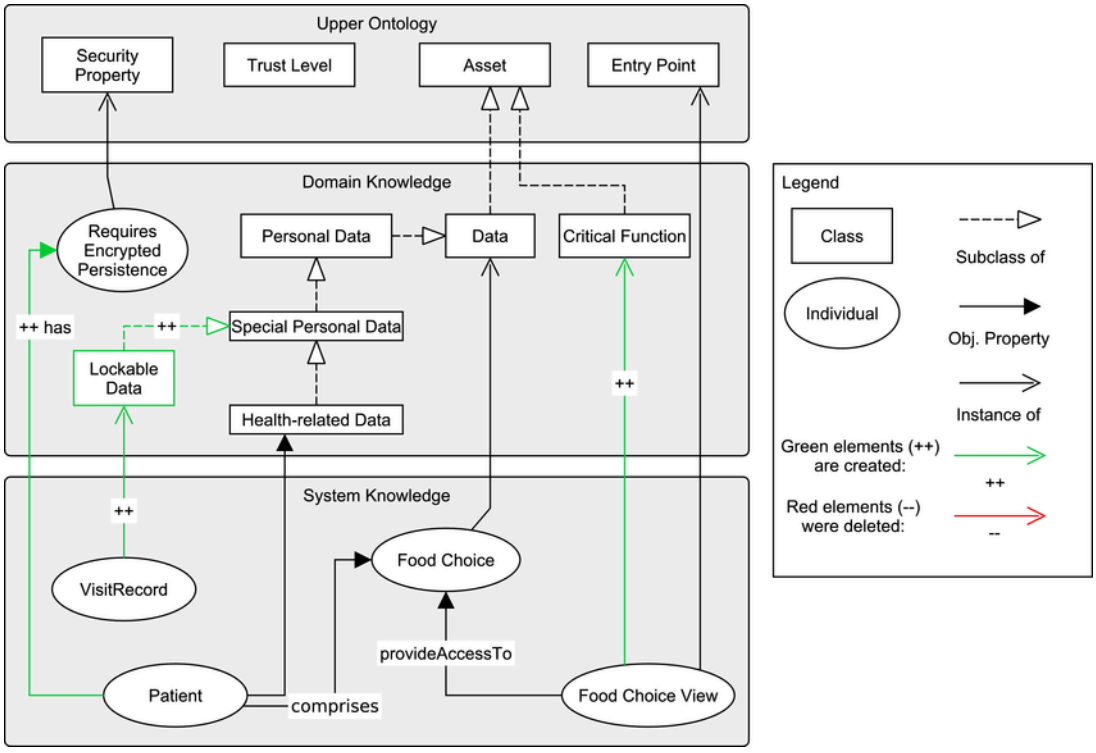

Fig. 11. SCK excerpt and changes regarding ESR3-5 (see Table 1).

Regarding ESR3, patients can have a food diary, in which they can view entries and edit food preferences. Thus, there is the asset *Food Choice* and the respective system component *Food Choice View*. As food habits can reveal also information about a persons' religion, viewing this information is newly considered to be critical and thus becomes an instance of this class.

Concerning ESR4, a system communicating via secure channels may still have an inadequate data integrity when storing data in an unencrypted manner. Here, the notions *Requires Encrypted Persistence* and *Critical Function* are provided by the CWE catalog.

Given the standard deployment of iTrust, it uses a SQL server (e.g. MySQL) to persist data assets. Thus, anybody who has access to the database server is able to gain access to all of the server's data easily. Given the evolution of the privacy regulations and further regulations for medical systems, it may be necessary to persist patient data using encryption. Thus, *Patient* now requires the respective security property.

Concerning ESR5, a change of the German Federal Data Protection Act in 2001 furnished patients with the right to lock certain data that are subject to automated processing, such as their visit records. In the domain ontology, we assume a class called *Lockable Data*, whose instances require the presence of appropriate locking mechanisms in the system.

4.3. Application of Security Maintenance Rules

Above, we presented five diverse knowledge changes. In our approach, we use Security Maintenance Rules to analyze the system model regarding these changes and co-evolve it where necessary. We now illustrate the application to the iTrust case study along the changes.

We reiterate ESR1 from Table 1, *Secure Encryption*, by taking into account the threat to the RC4 encryption algorithm introduced in Fig. 9. The knowledge change arrives in the form of an ontology diff, produced by the SiLift differencing tool by comparing the old and the evolved version of the overall ontology. Since the *ON* part of ESR1 matches this diff, our approach recognizes the need for co-evolution and triggers the corresponding Security Maintenance Rule from the table, which is applied to the system model.

Fig. 12 shows the relevant excerpt from iTrust's system model, a deployment diagram. To support the specification of security aspects in the deployment of the system, the diagram is equipped with the UMLsec stereotype ⟨⟨`secure links`⟩⟩. The corresponding stereotype ⟨⟨`encrypted enc`⟩⟩ is attached to each communication path between two nodes; it uses tagged values for specifying the encryption algorithm to be used as well as its key length (Jürjens, 2005). The tagged values are represented as annotations in the figure.

During the execution of the Security Maintenance Rule for ESR1 as per Table 1, the *IF* part is analyzed first. It requires that all `CommunicationPath` elements are examined regarding their annotation to gather insecure paths. As defined by *DO*, wherever RC4 is used, it is to be replaced by another, secure algorithm. The surrogate algorithm can either be inferred automatically from the knowledge or even further specified or approved by the user. In addition, a number of public institutions constantly provide publicly available recommendations for encryption techniques (BSI, 2012; NIST, 0000) which can also be incorporated into the Security Context Knowledge. The co-evolved deployment diagram is shown in Fig. 13. The vulnerable RC4 algorithm is now replaced by AES-256. It was chosen because it is the first result of the query to the Security Context Knowledge and the selection is shown to the user.

Focusing on ESR2 from Table 1, *No access to unauthorized users*, the knowledge evolution from Fig. 10 requires a reorganization of role access: the role *UAP* shall not be allowed to access the *Patient View* anymore. The *ON* part of ESR2 is triggered since a *accessibleBy* object property was removed.

The relevant system model excerpt, shown in Fig. 14, is a state chart specifying iTrust's functional behavior. It comprises use cases 3, 26 and 44, which describe the login into the system and the access to patient-specific information. The role access system is modeled using the UMLsec stereotype ⟨⟨`enusureRole`⟩⟩. This stereotype provides a tagged value to define the role a user needs to hold when trying to reach a specific state. Thus, ⟨⟨`ensureRole`⟩⟩ is annotated to state chart transitions. Unwanted behavior can be detected when a user holding an insufficient role tries to traverse an annotated transition.

To make iTrust compliant again, access for *UAP* needs to be restricted. The *IF* part initiates a model query to check if *UAP* has access to patient data. This is the case for the transition `OpenPatientInstructionDialog`: A path from the login using the parent role HCP to the specific transition can be traced. To this end, the state chart elements are traversed using a breadth first search.

In the *DO* part, the system behavior for the role `UAP` needs to be changed in a way that access to the critical transition `OpenPatientInstructionDialog` is removed. For all other roles, the system behavior remains unchanged.

This is realized by restricting access to the critical state (target of the above mentioned transition) by cloning the parent state `ModifyOV` first and removing the critical state. The user needs to provide

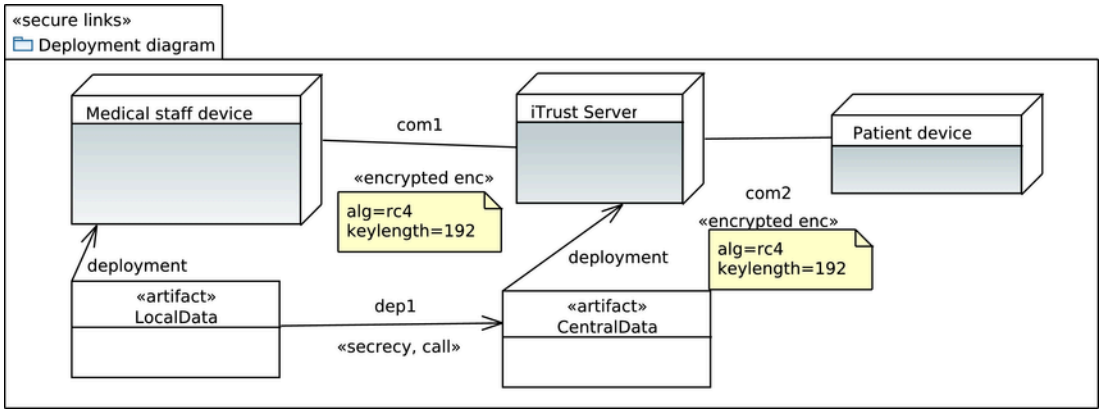

Fig. 12. Deployment diagram of iTrust before co-evolution.

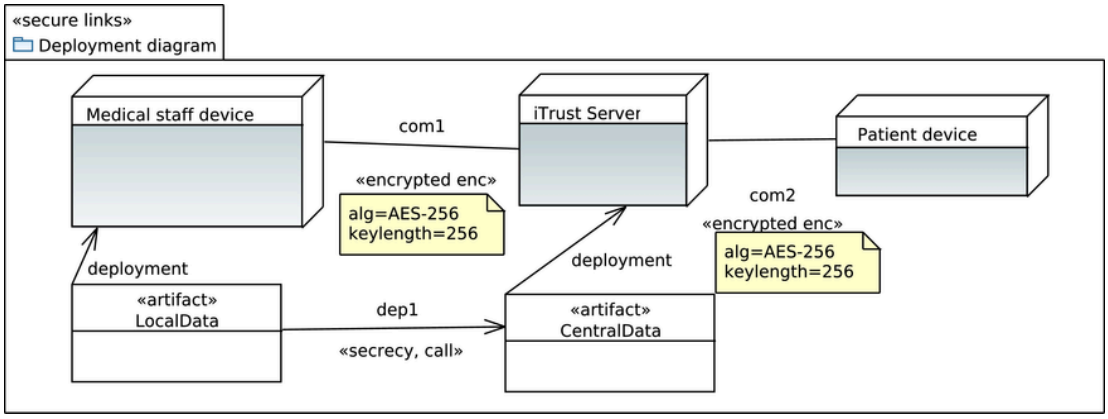

Fig. 13. Deployment diagram of iTrust after co-evolution.

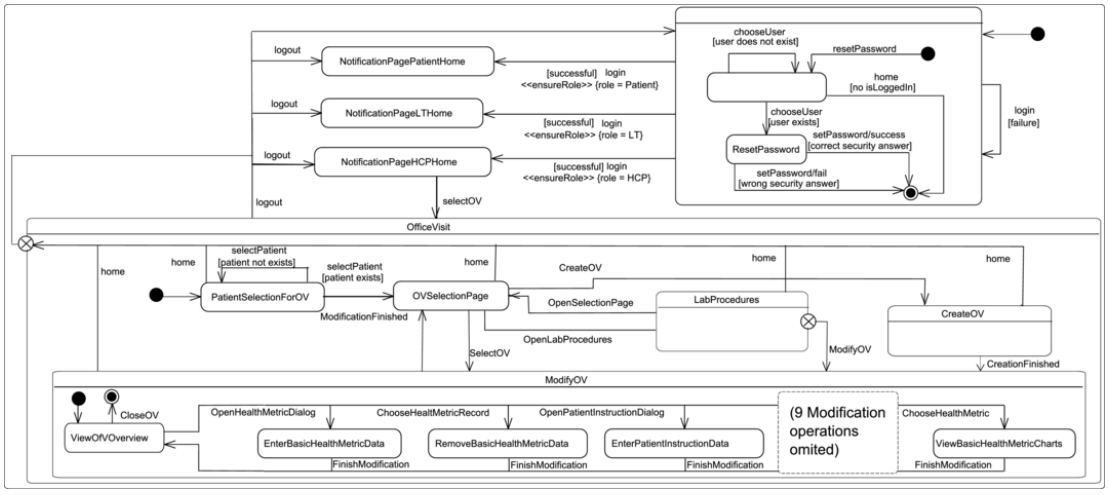

Fig. 14. Combined state chart showing the use cases 3, 26 and 44.

a name for the cloned complex state. The login state (begin of the path) has no `UAP` annotation, so this state is cloned, too, and assigned `UAP` as ⟨⟨`ensureRole`⟩⟩ tagged value. These changes are performed to the diagram shown in Fig. 14, re-establishing the compliance to the privacy requirements.

Regarding ESR3 from Table 1, *Authentication for critical functions in place*, we consider the knowledge evolution in Fig. 11. The *Food Choice View* has become a critical function, since it may support to infer a patient's religion. Consequently, the *ON* part of ESR3 is triggered because a link to the *Critical Function* class has been introduced.

IF initiates a model query and checks if the operation which provides access to the food choice view is secured. On the class diagram level, UMLsec provides the ⟨⟨`rabacRequire`⟩⟩ to specify access control according to the *role-centric attribute-based access control* model (RABAC, (Jin et al., 2012)). If an operation is tagged with this stereotype, it may only be executed if the current user has a role which is equipped with at least one right as defined in the tagged value `right`. As defined by *DO*, the stereotype is added automatically. Afterwards, the user is requested to provide a appropriate set of rights.

Fig. 15 presents a relevant excerpt from iTrust's system model: a class diagram excerpt portraying the `Patient` class. It shows the state after co-evolution and highlighted elements which have been

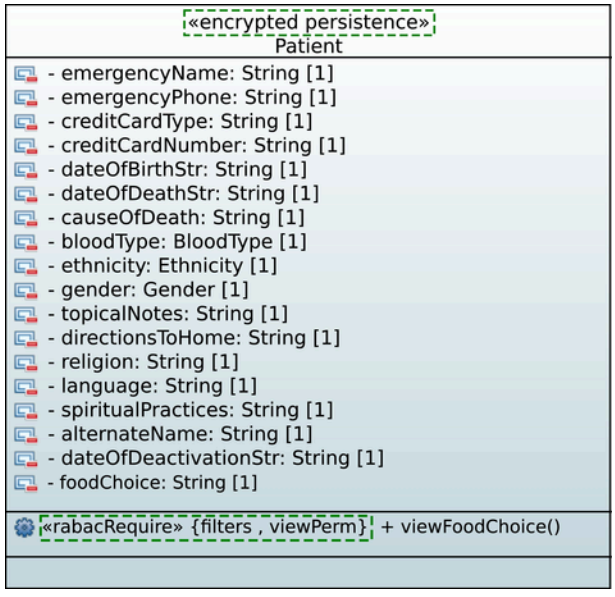

Fig. 15. Excerpt: `Patient` data class of iTrust.

added during the co-evolution. During the evaluation of the *IF* part, the lack of the required ⟨⟨`rabacRequire`⟩⟩ stereotype on the `viewFoodChoice` operation was observed. To ensure the enforcement of the required right, the stereotype was added in the *DO* part.

ESR4, *Encrypted Persistence*, is affected by `Encrypted Persistence` becoming mandatory for `Patients`, as shown in Fig. 11. The *ON* part of ESR4 is triggered by newly adding a connection between the *Encrypted Persistence* Class and an individual. Subsequently, the *IF* part checks if the respective model element `Patient` is equipped with a respective stereotype, and the *DO* part adds the stereotype where this applies. As presented in Fig. 15, UMLsec's stereotype ⟨⟨`encrypted persistence`⟩⟩ is used to tag data classes that are not allowed to be persisted without encryption.

Focusing on ESR5, *Locking is in place*, the German Federal Data Protection Act furnishes people the right to lock diagnosis-relevant data, which imposes particularly strict confidentiality requirements on visit reports. The corresponding Security Context Knowledge change involves the addition of a *requires* link between `VisitRecord` and `LockableData`. The *ON* part of ESR5 is designed to trigger the corresponding Security Maintenance Rule on such additions.

In consequence, the *IF* part of the rule checks if the affected data classes also have a `locked` property. If this is not the case, the application of *DO* leads to the addition of this property. The developer is informed to update the implementation so that it uses the property. The rule application works largely the same as in the above mentioned changes to Fig. 15.

4.4. Discussion

We carried out a case study by applying our approach to iTrust. As environment knowledge evolution scenario we used a newly introduced category of private data as happened to the German Federal Data Protection Act in 2001, as well as a typical scenario in software security engineering when a encryption algorithm is discovered to be insecure. We investigated five general security requirements as addressed by the community driven database of Common Weakness Enumerations (CWE).

Regarding ***RQ1***, we used our conception of layered ontologies to represent security knowledge. To this end, we complemented the upper ontology from Section 3.1 with domain and system ontologies, representing (1) domain knowledge about laws and security mechanisms how to achieve security properties, (2) relevant elements from the iTrust system to bridge the gap between knowledge modeling and the system model. As indicated by the examples in Section 4.2, we found that the graphical representation of ontologies supports gaining insight in crosscutting concepts of a system design like access control, role hierarchies, and security mechanisms used to implement security requirements of a system.

This knowledge can be modeled at a central place and uncluttered from more concrete system design details, which would be scattered over the system model or even not visible at all otherwise. On the downside, we found that working with two tools, Protégé and our own prototype, time-consuming. For providing system elements to the Security Context Knowledge, we started modeling system design elements manually and later on used a transformation which parses the system design model and integrates it into an ontology layer. A conceivable enhancement could be to build a plug-in for Protégé which lets one explore and link elements from the system design. A tighter integration or even building an ontology editor as part of our prototype would be useful to improve usability. However, re-implementing existing tooling is a large additional effort, while Protégé is well-known and widely used in the ontology community. Apart from these usability issues, modeling the knowledge and ontologies went out without further challenges, and it turned out be intuitive.

With regard to ***RQ2***, we used the concepts of Essential Security Requirements and Security Maintenance Rules, as introduced in Section 3.2. In all considered cases, our approach was able to detected requirement violations by utilizing the knowledge structure and graph expressions. Whenever a potential vulnerability was discovered, the Security Maintenance Rule investigated the system model by using model queries, eventually supported by Java code, to propose a co-evolution.

Essential Security Requirements, defining central security requirements of a system, were the starting point of our case study. As indicated by the examples in Section 5.4, we could define suitable Essential Security Requirements in all cases. Evolutions of the environmental knowledge were detected using SiLift, triggering the application of the corresponding Security Maintenance Rules. SiLift supports our approach in bridging the gap between ontology-based knowledge modeling and the UML-based system modeling.

We found that the Henshin graph transformation language is useful for expressing most of the required rules in a declarative and intuitive manner. First, model queries can be modeled by using simply ⟨⟨`preserve`⟩⟩ nodes in the transformation rules. Due to its unified view on transformations by displaying LHS and RHS in one view, the user does not experience doubling of rule nodes which keeps the user interface clear. This is especially helpful when modeling complex rules. Using Henshin in conjunction with its auxiliary mechanism partial match, co-evolution steps can also be kept simple. To alter model elements in a definite scope of the model, there is no need to provide rules which model the whole containment hierarchy up to the model's root element. It will suffice to focus on the directly related elements and provide a partial match for a number of *anchor* elements to support the matcher enough to find the desired spot in the model.

Additionally, Henshin also serves as the *user interface* of the SiLift tool. Semantic differences which can be processed by SiLift are modeled using Henshin transformation rules. This keeps the learning curve flat.

For example, Fig. 16 depicts a rule we use for co-evolution to add an UMLsec annotation ⟨⟨`ensureRole`⟩⟩ to a transition in a state chart. By utilizing Henshin's partial match mechanism, we can reuse knowledge already acquired by preceding steps, provide the concrete `Transition` instance as and can directly apply the rule.

In a few cases, we had to surpass expressiveness restrictions of Henshin by involving a moderate amount of supporting Java code. Examples include path expressions, and cloning of model parts with a containment hierarchy, we accompanied graph transformation rules with it.

We found that using UMLsec stereotypes provide a lightweight possibility of annotating concrete parts of the system design with supplementary details concerning security requirements, like restrictions

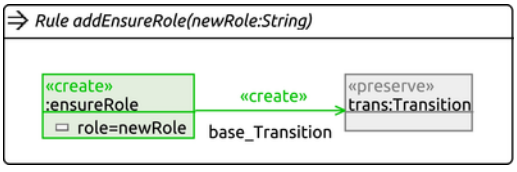

Fig. 16. Example of Henshin transformation rule minimized due to use of partial match mechanism.

for accessing certain methods or states of a system. The annotations integrate into the existing UML models and extend the syntax via the officially provided profile mechanism. Designing the models is not hampered because all model design features work like before and are just enhanced by the new aspects. If necessary, a view on the model without annotations can be generated easily e.g. by temporarily remove every application of the UMLsec profile.

Regarding ***RQ3***, we presented co-evolutions as carried out by Security Maintenance Rules (SMR). Co-evolution steps can be applied automatically in case there is a single way to recover the ESR. In other cases, the user can provide additional information. For example, we found modeling Security Maintenance Rule for ESR1 (see Fig. 9) quite straightforward. It required least user interaction because all parts of the system to be altered can be directly inferred by the knowledge change. In fact, co-evolution can be applied directly by choosing a now secure algorithm and apply it to the whole system design, wherever the now insecure algorithm is referred.

Modeling the Security Maintenance Rule for ESR2 (see Fig. 10) proved to require the most manual effort. As can be already seen in comparison to Fig. 9, ESR2 interacts with a number of concepts of the concrete system. This comprises the role model as well as parts of the system like the *Patient View* user interface element and *Patient* data object. The synchronization of knowledge by using a heuristics-supported round-trip approach could lower the manual effort here. By applying these co-evolutions, the given system model has UMLsec security annotations that are compliant with the current context knowledge again. Updates to the system model still have to be propagated to the system implementation. In some cases, an automated generation of the corresponding code is possible; in other cases, additional manual implementation effort is required.

In summary, we found that the modularized modeling of the system, knowledge and security aspects lead to more clarity. Our semi-automatic approach eases co-evolving a system design when knowledge changes come into place because it minimizes effort of inspecting artifacts manually, which especially gets more useful the bigger the considered system is. With our contribution, we support the maintenance of long-living systems.

4.5. Threats to validity

According to Wohlin et al.'s taxonomy of threats to validity (Wohlin et al., 2012), *external validity* is key in applied research: results should generalize to other systems of a particular size or application domain. Our two main threats for external validity concern the considered system and participants in our case study.

The considered system, iTrust, comes from an academic setting rather than from an industry partner. In Section 4.1, we argue that it is representative for systems in industry, since several key properties of long-living, complex, and security-critical systems apply. In the future, we plan to use our approach in industrial systems as well.

The independent team responsible for creating the involved models were academics. To mitigate the associated threat, the team was composed carefully: the team leader was equipped with ten years of industrial development experience, and the full team was composed of persons with development experience as well. Still, an empirical evaluation in the field is left as future work.

Internal validity refers to the validity of causal relationship between treatment and outcome. Since our subject system is from the domain of long-living systems, it may be subject to *maturation*: As time has passed, available weaknesses might have been mitigated, preventing us from detecting and fixing them. Our set-up focuses on selected weaknesses, in which our approach showed to be helpful: it was able to support the semi-automated co-evolution of the system.

Construct validity focuses on the relation between theory and observation. We operationalized the considered construct, usefulness, as the ability to support semi-automated co-evolution. This operationalization does not focus on the usefulness concern of *effort*, which would require a quantitative evaluation setup. However, all prime benefits of our approach are related to the reduction of effort: Representing security knowledge in a central knowledge base reduces redundant communication effort. Model queries reduce manual analysis effort. Our semi-automated co-evolution reduces manual maintenance effort.

5. Related work

In this section we give an overview of publications related to our approach. As our approach covers several areas of research, we have structured the related work we discuss according to the respective main topics.

5.1. Techniques to support security requirements elicitation

In the last decade, several methods and tools have been developed to create threat models feasible to identify and mitigate threats (cf. Myagmar et al. (2005); Oladimeji et al. (2006)). Moreover, a few approaches consider use cases as a starting point to identify threats as a basis for security requirements.

Sindre and Opdahl (2005) argue that use cases offer limited support for eliciting security requirements and therefore regard use cases enriched with textual description about misuse. Based on this, guidelines are presented how to describe misuse cases in detail, so that, in a next step, method guidelines for eliciting security requirements with misuse cases can be established. While providing a general approach on how to elicit security requirements based on use case descriptions, in contrast to our approach there is no focus on overall security goals / requirements and relations to the later system design.

Kaiya et al. (2013) use information about the underlying architecture to elicit security requirements. Use-case descriptions are converted into data-flow diagrams called asset flow diagrams. Security requirements are then defined as countermeasures for an attack if detected by the asset flow diagrams and as design and implementation constraints if not detected. In contrast to the SecVolution approach, there is no systematic approach for evolving stakeholder needs.

Haley et al. (2008) present a framework not only for security requirements elicitation, but also for security analysis. Their method is based on constructing a context for the regarded system. Describing this context with a problem oriented notation makes it possible to validate the system against the security requirements. The approach is very powerful but needs a lot of security expertise to build the context and understand the results of the analysis. Evolution of the context is not supported here.

5.2 Security patterns incorporating requirements elicitation.Security patterns in frontage of Requirements Elicitation

Harmain and Gaizauskas (2003) utilize Natural Language Processing (NLP) for eliciting software requirements. With their technique they are able to detect entities related to classes and attributes to build an initial UML class model. In contrast to our approach they are not considering requirements in context of security issues.

Compagna et al. (2008) integrate legal patterns into a requirements engineering methodology for the development of security and

privacy patterns using NLP. This description is parsed by a natural language processor on the basis of a semantic template. The pattern design and validation process requires legal experts to describe patterns in natural language. While providing a possibility to model entities and their security needs in an SI* model, there is no process provided how to hand the knowledge over to subsequent development steps.

Gegick and Williams (2007) developed a methodology for early identification of system vulnerabilities for existing threats based on regular expressions. Patterns of possible vulnerabilities are used to identify threats. The method is called Security Analysis for Existing Threats (SAFE-T) and is not tailored to deal with newly discovered thready and a system that is deployed.

5.3Managing security knowledge to appropriately react to evolution.Managing Security Knowledge to adequate react to Evolution

Tsoumas and Gritzalis (2006) provide a security ontology based framework for enterprises linking high level policy statements and deployable security controls. The security ontology is build by extending the DMTF Common Information Model standard. It is used as a container for the security related information that concerns the information system. In contrast to SecVolution, this approach targets organizational security controls (e.g. securing server hardware, recommending using a firewall) and not developing secure software systems.

Ernst et al. (2011) identified changes in requirements specification as triggering event for software evolution. The relationship between requirements and implementation is described formally. Together with goals derived from software specification, implementation tasks are computed to reach the goals in accordance with the requirements. The goal of this formal, rather abstract approach is to provide a clearly structured, graph-based guidance for implementation decisions. A co-evolution scenario arises when changing requirements restate the requirements problem, which the authors state is not solved yet. An interface to system design level is not discussed.

Souza et al. (2013) regard requirements that cause the evolution of other requirements. Their approach is based on goal-oriented modeling. The system the stakeholders expect is supposed to be modeled as a set of goals. During run time, events can be monitored. The concept of evolution requirements (EvoReqs) based on event-condition-action schema, is used to adapt the goal model. Security is only considered as a side note. The adaption capabilities are fully based on the rules as provided upfront and as part of the closed system, there is no mechanism of accessing knowledge like general security knowledge.

Salehie et al. (2012) present a requirements driven approach to adaptive security which aims at identifying and evaluating changing assets at run time to dynamically enable different countermeasures. A casual network is build upon a goal model and a threat model to analyze system security in different situations. The causal network needs to be maintained manually and dealing with unanticipated events is not covered.

The systematic investigation of evolution of the software system when security knowledge in the environment changes is a topic of ongoing research. With our holistic approach the security engineer is able to concentrate on finding new threats and attack patterns rather than dealing with known threats.

5.4. Analyze the impact of changes with respect to co-evolution

When an evolution took place, the software artifacts under consideration could be inconsistent due to unintentional side effects.

The *Water wave phenomenon* inspired Li et al., 2013) to develop an impact assessment approach based on call graphs. First they analyze the core, which consists of the direct affected software artifacts. After that, the call graph is analyzed, taking the interference of different changes into account. They claim that their nature inspired approach has fewer false positives compared to common call-graph approaches. Their approach is focused on predicting how big (i.e. number of methods to change) the impact of changing a number of methods in a given source code project will be. Opposed to this, our approach aims at analyzing impact regarding security properties.

The main idea of Bouneffa and Ahmad (2014) is to use semantic knowledge about artifacts and change operations that is represented in an ontology to realize a change management. Change operations are formalized as graph rewriting rules and implement the change and its impact propagation. The approach does not support processing of requirements or other overall properties. The change management presented instead serves for keeping a big model consistent.

Okubo et al. (2011) regard the impact of software enhancements on security by involving patterns of enhancements. The overall goal is to enable the developer to estimate and compare the amount of modifications needed by different countermeasures. The proposed security analysis process uses security requirements patterns to identify threats and security design patterns to find countermeasures (see also Okubo et al., 2012). In contrast to SecVolution, their approach does not cover evolution. Besides, application of the approach leads to having a guidance of which patterns for security precautions or vulnerability mitigations should be applied. There is no integration with an actual system design so far.

5.5. Vulnerability and attack management

A main type of security threats arises from the constant emergence of new vulnerabilities and corresponding attacks.

An investigation of Kuehn and Mueller (2014) focuses on zero-day exploits, in which previously unknown vulnerabilities are exploited by attackers. Big players such as Microsoft or Facebook face a desperate situation, where conventional security precautions seem to be overwhelmed by a rapidly increasing number of these exploits. However, rather than addressing the question how vulnerabilities can be avoided upfront, their current reaction is to take part in the race by conducting bug bounty programs. In contrast, SecVolution builds upon community knowledge. Using layered ontologies, knowledge can be hierarchically structured. As soon as an attack is discovered (i.e. knowledge about it is made explicit), it can be shared publicly, which speeds up the vulnerability fixing process.

Alhazmi et al. (2007) define a metric called *vulnerability density*, which puts the number of vulnerabilities of a product in relation to its overall bug count. They define a logistic and a linear model and measure its fitness regarding the publicly available bug data of a number of Microsoft Windows releases and two Red Hat Linux releases.

The authors state that their metric can be used to predict the number of vulnerabilities to be expected. This method could be useful for our approach to gain additional knowledge and pointing out potential system parts which may be vulnerable.

Since SecVolution focuses on the system level, there is a trade-off regarding the granularity of vulnerabilities and attack types it can address. We focus on design-level vulnerabilities, such as the use of an

outdated encryption algorithm in our case study, or a version of the OpenSSL library affected by the Heartbleed bug. Implementation-level vulnerabilities such as buffer overflows must be detected and fixed on the source-code level, for which a plethora of tools exist.

6. Conclusion and future research

Maintaining long-living information systems regarding security by taking a changing environment into account is a challenging task. Security-related parts of a software have to be changed when environmental knowledge changes or when assumptions about the system context do not hold any longer due to occurred security incidents.

In this paper, we presented an approach to model environmental knowledge, cope with its evolution and co-evolve the system model of an information system semi-automatically. The presented work is part of our SecVolution approach that is intended to retain security of long-living information systems independent of the process model, domain, or technology in use.

The presented approach determines the knowledge evolution impact using the given security requirements.

We further presented how the Essential Security Requirements are encoded to query the system model. This is important to highlight parts of the system model that do not fulfill a particular requirement anymore. Based on this information, appropriate adaptations are selected to co-evolve the system model.

In the case study, we have shown that our approach is applicable and at least reduces the effort for security experts to identify vulnerable portions of the system model and to avoid conventional security problems automatically. Although we only focus on privacy aspects within the case study, our approach is capable to deal with other aspects (e.g. changing technology, occurred security incidents) as well.

6.1. Future research

The work presented in this article is part of ongoing research. The work on the project SecVolution is extended as part of a second phase of the priority programme. By now, our approach focused on a development cycle consisting of design phase and implementation. Due to enterprise service buses and further approaches, where different components are integrated to deliver a more complex service, as well as on-line services which ought to be used 24/7 by mobile devices, maintenance cycles with a substantial downtime to re-run the development cycle often cannot be tolerated. For example, an unwanted role/rights configuration or system behavior specified at design-time should be monitored and (in critical cases) acted upon during system execution. Thus, in our ongoing research, we will also focus on the deployment and maintenance phase of information systems.

The SecVolution approach currently is designed to support development cycles and projects by gradually wrapping an existing system into a layer of security-related knowledge because it tends to support long-living information systems.

Thus, the system can only be enhanced by monitoring probes to a certain extent that generate run-time data. Our future research will focus on determine the need for adaptation at run-time based on monitoring and system evolution.

Moreover, we will conduct research on how to preserve compliance to security requirements by run-time adaptation based on security knowledge and its evolution. It has to be decided if a security issue has to be mitigated by using run-time adaptation reactions, if a more in-depth determining is necessary in design-time by a re-run of the development cycle or if a temporary take-down of functionalities is acceptable. Moreover, ordinary requirements should not be affected as well as it has to be prevented that the system slowly degrades by continuous adaptation applications. This is a multi-objective optimization problem.

Acknowledgment

This research is funded by the DFG project SecVolution@Run-time (JU 2734/2-2 and SCHN 1072/4-2) which is part of the priority programme SPP 1593 “Design For Future - Managed Software Evolution”.

References

Ahmadian, A.S., Peldszus, S., Ramadan, Q., Jürjens, J., 2017. Model-based privacy and security analysis with CARiSMA. Joint Meeting on Foundations of Software Engineering (ESEC/FSE). 989–993.

Alhazmi, O.H., Malaiya, Y.K., Ray, I., 2007. Measuring, analyzing and predicting security vulnerabilities in software systems. Comput. Secur. 26 (3), 219–228.

Arendt, T., Biermann, E., Jurack, S., Krause, C., Taentzer, G., 2010. Henshin: Advanced concepts and tools for in-place EMF model transformations. Model Driven Engineering Languages and Systems (MoDELS). 121–135.

Baras, D.S.A., Othman, S.H., Ahmad, M.N., Ithnin, N., 2014. Towards managing information security knowledge through metamodelling approach. International Symposium on Biometrics and Security Technologies (ISBAST). IEEE, 310–315.

Bouneffa, M., Ahmad, A., 2014. The change impact analysis in BPM based software applications: a graph rewriting and ontology based approach. Enterprise Information Systems. Springer, 280–295.

Bowman, S., 2013. Impact of electronic health record systems on information integrity: quality and safety implications. Perspect. Health Inf. Manag. 10. Fall.

Braz, F.a., Fernandez, E.B., VanHilst, M., 2008. Eliciting Security Requirements through Misuse Activities. International Workshop on Secure Systems Methodologies Using Patterns (SPattern). IEEE, 328–333.

Brereton, P., Kitchenham, B.A., Budgen, D., Turner, M., Khalil, M., 2007. Lessons from applying the systematic literature review process within the software engineering domain. J. Syst. Softw. 80 (4), 571–583.

BSI, 2012. IT Basic Protection Catalog.

Bundesamt für Datenschutz, 2014. BDSG Änderungen - Änderungen des Bundesdatenschutzgesetzes seit 1990.

Bundesministerium des Inneren, 2005. Bundesdatenschutzgesetz. Bundesgesetzblatt.

Bürger, J., Gärtner, S., Ruhroth, T., Zweihoff, J., Jürjens, J., Schneider, K., 2015. Restoring security of long-living systems by co-evolution. International Computer Software and Applications Conference (COMPSAC). 153–158.

Bürger, J., Jürjens, J., Ruhroth, T., Gärtner, S., Schneider, K., 2014. Model-based Security Engineering with UML: managed co-evolution of Security Knowledge and Software Models. Foundations of Security Analysis and Desing VII: FOSAD Tutorial Lectures. Springer, 34–53.

Bürger, J., Jürjens, J., Wenzel, S., 2015. Restoring security of evolving software models using graph transformation. Int. J. Softw. Tools Technol. Transf. 17 (3), 267–289.

Compagna, L., El Khoury, P., Krausová, A., Massacci, F., Zannone, N., 2008. How to integrate legal requirements into a requirements engineering methodology for the development of security and privacy patterns. Artif. Intell. Law 17 (1), 1–30.

Dayal, U., 1994. Active Database Systems: Triggers and Rules for Advanced Database Processing. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

Dubois, , Heymans, P., Mayer, N., Matulevičius, R., 2010. A systematic approach to define the domain of information system security risk management. Intentional Perspectives on Information Systems Engineering. Springer, 289–306.

Eclipse Foundation,. Eclipse modeling framework project (EMF).

Elahi, G., Yu, E., Zannone, N., 2009. A vulnerability-centric requirements engineering framework: analyzing security attacks, countermeasures, and requirements based on vulnerabilities. Requirements Eng. 15 (1), 41–62.

Ernst, N.A., Borgida, A., Mylopoulos, J., 2011. Requirements evolution drives software evolution. International Workshop on Principles of Software Evolution and ERCIM Workshop on Software Evolution. ACM, 16–20.

Gärtner, S., Ruhroth, T., Bürger, J., Schneider, K., Jürjens, J., 2014. Maintaining requirements for long-living software systems by incorporating security knowledge. International Requirements Engineering Conference (RE). IEEE, 103–112.

Gegick, M., Williams, L., 2007. On the design of more secure software-intensive systems by use of attack patterns. Inf. Softw. Technol. 49 (4), 381–397.

Gruber, T.R., 1995. Toward principles for the design of ontologies used for knowledge sharing?. Int. J. Hum. Comput. Stud. 43 (5–6), 907–928.

Habela, P., Kaczmarski, K., Stencel, K., Subieta, K., 2008. OCL as the query language for UML model execution. International Conference on Computational Science (ICCS). Vol. 5103, Springer Berlin Heidelberg, 311–320.

Haley, C.B., Laney, R.C., Moffett, J.D., Nuseibeh, B., 2008. Security requirements engineering: a framework for representation and analysis. IEEE Trans. Software Eng. 34 (1), 133–153.
Harmain, H., Gaizauskas, R., 2003. CM-Builder: A natural language-based case tool for object-oriented analysis. Autom. Softw. Eng. 10, 157–181.
Houmb, S.H., Islam, S., Knauss, E., Jürjens, J., Schneider, K., 2009. Eliciting security requirements and tracing them to design: an integration of common criteria, heuristics, and UMLsec. Requirements Eng. 15 (1), 63–93.
Hutchinson, J., Whittle, J., Rouncefield, M., 2014. Model-driven engineering practices in industry: social, organizational and managerial factors that lead to success or failure. Sci. Comput. Program. 89, 144–161.
International Standardization Organization, 2007. ISO 15408:2007 Common Criteria for Information Technology Security Evaluation, Version 3.1, Revision 2, CCMB-2007-09-001, CCMB-2007-09-002 and CCMB-2007-09-003.
Jhawar, R., Kordy, B., Mauw, S., Radomirović, S., Trujillo-Rasua, R., 2015. Attack trees with sequential conjunction. IFIP International Information Security Conference (SEC). Springer, 339–353.
Jin, X., Sandhu, R., Krishnan, R., 2012. RABAC: Role-Centric Attribute-Based Access Control. Springer, 84–96.
Jürjens, J., 2005. Secure Systems Development with UML. Springer.
Jürjens, J., Schneider, K., 2014. The SecReq approach: from security requirements to secure design while managing software evolution. Software Engineering (SE). GI, 89–90.
Kaiya, H., Sakai, J., Ogata, S., Kaijiri, K., 2013. Eliciting security requirements for an information system using asset flows and processor deployment. Int. J. Secure Softw. Eng. 4 (3), 42–63.
Kehrer, T., Kelter, U., Ohrndorf, M., Sollbach, T., 2012. Understanding model evolution through semantically lifting model differences with SiLift. International Conference on Software Maintenance (ICSM). 638–641.
Kitchenham, B., Charters, S., 2007. Guidelines for performing systematic literature reviews in software engineering. Technical Report. EBSE-2007-01.
Knauss, E., Lübke, D., Meyer, S., 2009. Feedback-driven requirements engineering: the Heuristic requirements assistant. International Conference on Software Engineering (ICSE). 587–590.
Kuehn, A., Mueller, M., 2014. Shifts in the cybersecurity paradigm: Zero-day exploits, discourse, and emerging institutions. New Security Paradigms Workshop (NSPW). ACM, 63–68.
Launders, I., Polovina, S., 2013. A semantic approach to security policy reasoning. Strategic Intelligence Management. Elsevier, 150–166.
Lehman, M.M., Ramil, J.F., 2003. Software evolution – background, theory, practice. Inf. Process. Lett. 88 (1–2), 33–44.
Li, B., Zhang, Q., Sun, X., Leung, H., 2013. Using water wave propagation phenomenon to study software change impact analysis. Adv. Eng. Softw. 58, 45–53.
McManus, H., Hastings, D., 2005. A framework for understanding uncertainty and its mitigation and exploitation in complex systems. INCOSE International Symposium. Vol. 15, Wiley Online Library, 484–503.
Meneely, A., Smith, B., Williams, L., 2012. iTrust electronic health care system case study. Software and Systems Traceability. Springer, 425–438.
Miede, A., Nedyalkov, N., Gottron, C., König, A., Repp, N., Steinmetz, R., 2010. A generic metamodel for it security attack modeling for distributed systems. International Conference on Availability, Reliability, and Security (ACES). IEEE, 430–437.
Mitleton-Kelly, E., Papaefthimiou, M.-C., 2002. Co-evolution of Diverse Elements Interacting Within a Social Ecosystem. Springer.
MITRE, 2013. Common Vulnerabilities and Exposures.
MITRE, 2017. Common Weakness Enumeration.
Mouratidis, H., Giorgini, P., Manson, G., 2003. An ontology for modelling security: the tropos approach. International Conference on Knowledge-Based and Intelligent Information and Engineering Systems (KES). Springer, 1387–1394.
Myagmar, S., Lee, A.J., Yurcik, W., 2005. Threat modeling as a basis for security requirements. IEEE Symposium on Requirements Engineering for Information Security (SREIS). 1–8.
NIST,. Recommendation for Key Management. NIST Special Publication 800-57 Part 1 Revision 4.
Okubo, T., Kaiya, H., Yoshioka, N., 2011. Effective security impact analysis with patterns for software enhancement. Int. Conf. Availab. Reliab. Security 527–534.
Okubo, T., Kaiya, H., Yoshioka, N., 2012. Analyzing impacts on software enhancement caused by security design alternatives with patterns. Int. J. Secure Softw. Eng. 3 (1), 37–61.
Oladimeji, E., Supakkul, S., Chung, L., 2006. Security threat modeling and analysis: A goal-oriented approach. IASTED International Conference on Software Engineering and Applications (SEA). 13–15.
Realsearch Research Group North Carolina State University,. iTrust wiki. Accessed Dec 20, 2017.
RFC, 2015. RFC 7465: Prohibiting RC4 cipher suite.
Robillard, P., 1999. The role of knowledge in software development. Commun. ACM 42 (1), 87–92.
Ruhroth, T., Gärtner, S., Bürger, J., Jürjens, J., Schneider, K., 2014. Towards adaptation and evolution of domain-specific knowledge for maintaining secure systems. International Conference on Product Focused Software Process Improvement (PROFES). 239–253.
Salehie, M., Pasquale, L., Omoronyia, I., Ali, R., Nuseibeh, B., 2012. Requirements-driven adaptive security: protecting variable assets at runtime. International Requirements Engineering Conference (RE). IEEE, 111–120.
Schneider, K., Knauss, E., Houmb, S., Islam, S., Jürjens, J., 2011. Enhancing security requirements engineering by organizational learning. Requirements Eng. 17 (1), 35–56.
Schwaber, K., Sutherland, J., 2017. The Scrum Guide. Accessed Dec 20
Sindre, G., Opdahl, A.L., 2005. Eliciting security requirements with misuse cases. Requirements Eng. 10 (1), 34–44.
Souza, V.E.S., Lapouchnian, A., Angelopoulos, K., Mylopoulos, J., 2013. Requirements-driven software evolution. Comput. Sci.-Res. Develop. 28 (4), 311–329.
Stanford Center for Biomedical Informatics Research (BMIR),. Protégé homepage. http://protege.stanford.edu.
Swiderski, F., Snyder, W., 2004. Threat Modeling. Microsoft Press.
Tsoumas, B., Gritzalis, D., 2006. Towards an ontology-based security management. 20th International Conference on Advanced Information Networking and Applications (AINA). IEEE, 985–992.
Ujhelyi, Z., Bergmann, G., Ábel Hegedüs, Ákos Horváth, Izsó, B., Ráth, I., Szatmári, Z., Varró, D., 2015. EMF-IncQuery: an integrated development environment for live model queries. Sci. Comput. Program. 98, Part 1, 80–99.
Vanhoef, M., Piessens, F., 2015. All your biases belong to us: Breaking RC4 in WPA-TKIP and TLS. USENIX Security Symposium, USENIX Security 15. 97–112.
Wohlin, C., Runeson, P., Höst, M., Ohlsson, M.C., Regnell, B., Wesslén, A., 2012. Experimentation in Software Engineering. Springer Science & Business Media.