# Maintaining Requirements for Long-Living Software Systems by Incorporating Security Knowledge

Stefan Gärtner*, Thomas Ruhroth†, Jens Bürger†, Kurt Schneider*, and Jan Jürjens†
* Software Engineering Group, Leibniz Universität Hannover, Germany
{stefan.gaertner,kurt.schneider}@inf.uni-hannover.de
†Chair of Software Engineering, TU Dortmund, Germany
{thomas.ruhroth,jens.buerger,jan.jurjens}@cs.tu-dortmund.de

***Abstract*—Security is an increasingly important quality facet in modern information systems and needs to be retained. Due to a constantly changing environment, long-living software systems "age" not by wearing out, but by failing to keep up-to-date with their environment. The problem is that requirements engineers usually do not have a complete overview of the security-related knowledge necessary to retain security of long-living software systems. This includes security standards, principles and guidelines as well as reported security incidents. In this paper, we focus on the identification of known vulnerabilities (and their variations) in natural-language requirements by leveraging security knowledge. For this purpose, we present an integrative security knowledge model and a heuristic method to detect vulnerabilities in requirements based on reported security incidents. To support knowledge evolution, we further propose a method based on natural language analysis to refine and to adapt security knowledge. Our evaluation indicates that the proposed assessment approach detects vulnerable requirements more reliable than other methods (Bayes, SVM, k-NN). Thus, requirements engineers can react faster and more effectively to a changing environment that has an impact on the desired security level of the information system.**

***Index Terms*—Security requirements, Heuristics, Requirements analysis, Software evolution, Knowledge carrying software**

## I. Introduction

Security is an important quality facet in modern information systems and has to be particularly considered when maintaining such systems. But changing environmental conditions as well as growing attacker knowledge can compromise security of long-living information systems, even if the system itself remains unchanged. As a consequence, a system must be continuously protected against security breaches in a proactive manner. Moreover, software systems tend to degrade when they are modified and extended over a long period of time. Adding a new functionality to a system or modifying one may introduce security-related data and features which remain uninspected. Furthermore, security incidents which occur over time may lead to changes of various parts of the system. These changes may introduce serious vulnerabilities to the regarded information system. The challenge is to maintain an appropriate level of security over a long period of time.

To address this problem, we proposed the SecVolution approach in [34], which continuously monitors security knowledge and its evolution in order to determine required changes to the information system. It therefore combines heuristics for identifying security issues in requirements with a security preservation technique focusing on the system model. They are integrated in a learning cycle supported by knowledge of the environment. The knowledge supports identification of recurring problems in incoming changes. Whenever a problem is detected, a co-evolution mechanism provides semiautomatic recovery of the desired security level of the system model. By applying our SecVolution approach, a long-living system is wrapped into a layer of relevant security knowledge.

Assessing requirements of an information system with respect to security is worthwhile because software design is directly affected by requirements. Considering security later or not at all may result in an insecure system that leads to higher maintenance costs. In this paper, we therefore partly apply our SecVolution approach to support security assessment of requirements. Regarding long-living information systems, security knowledge and its evolution with respect to given requirements is focused. Thus, we propose an integrative security knowledge model to enable the reuse of knowledge. Furthermore, we present a heuristic method that uses this knowledge to support identification of security vulnerabilities in natural language requirements.

The goal of our research is not to detect or even to forecast new security issues, but to incorporate security knowledge in form of reported security incidents supporting security assessment of requirements. Therefore, we focus on requirements specifications by means of use cases. They support understanding of what the requirements of the system are and how the system works.

The remainder of this paper is structured as follows. In Sec. II, we sketch our approach and introduce our research questions. In Sec. III, we review existing security taxonomies and ontologies to identify relevant concepts of security knowledge. Based on the security knowledge, a security requirements assessment built upon heuristics is introduced in Sec. IV and details are presented in Sec. V. In order to refine and to extend the security knowledge, we address the extraction of additional knowledge based on heuristic findings in Sec. VI.

978-1-4799-3032-6/14/$31.00 © 2014 IEEE

RE 2014, Karlskrona, Sweden

In Sec. VII, the proposed assessment approach is evaluated by using the iTrust case study. Related research is presented in Sec. VIII. Finally, we conclude our work and outline future research in Sec. IX.

## II. Research Objective

This paper addresses the management and usage of evolving security knowledge in requirements engineering. The benefit of our approach is to support requirements engineers, so that they are able to invest their efforts into prevention and security improvement instead of finding known security issues in lengthy specification documents.

To achieve this objective, we propose an approach as presented in Fig. 1. Our approach consists of two parts: (1) Security Assessment and (2) Security Knowledge Extraction. In the first part, a given specification is assessed using heuristics and security knowledge. Heuristic findings are passed to the second part where a requirements engineer examines them and extracts additional knowledge as well as security requirements. Additional knowledge is used to adapt or to refine applied security knowledge whereas security requirements are passed to subsequent development activities. Regarding our approach, we have to face three research questions as depicted in Fig. 1.

Security knowledge is spread among information sources of very different kind (e.g. laws and regulations, attackers and white hats, incidents, developers, other stakeholders). Even security experts cannot entirely overview the relevant knowledge. Thus, we need to know ***(RQ1:)*** *How to organize security knowledge in a way that it can be used for assessing requirements of a long-living software system?*

To offer useful support for requirements engineers based on reported security incidents, we need to know ***(RQ2:)*** *How can requirements engineers identify security-critical issues in natural language requirements semiautomatically?* Here, we focus on use cases which need to be transformed into an appropriate analysis model applying natural language processing. After identifying security issue candidates, requirements engineers need to spot the candidates which really threaten the system.

The core concept of our approach is to use security knowledge modeled beforehand. However, modeling a huge amount of knowledge is a labor-intensive task. This leads to ***(RQ3:)*** *How can requirements engineers be supported to extract proper security knowledge from identified security-critical issues in requirements?* Since the extraction of knowledge depends on the way the knowledge is organized, this question is related with our first research question.

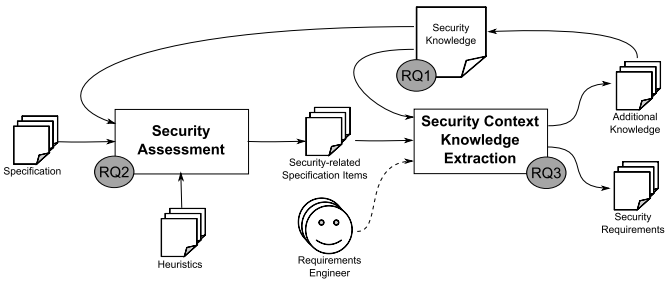


Fig. 1. Overview of the proposed approach including research questions.

To illustrate our concepts and to evaluate our approach, we choose the project iTrust [27] as case study. It is a medical application which supports patients to manage their health records as well as medical staff to organize their work. The health care sector is quite complex and security plays an important role. For example, a lot of sensitive patient data have to be stored. The iTrust project was founded at North Carolina State University and is currently maintained by the Realsearch Research Group [27]. It consists of 55 use cases written in natural language (version 23). Based on these requirements, the iTrust system (version 17) has been developed as a web application. Thus, the listed requirements are sufficient to develop a working system.

## III. Modeling Security Knowledge

Security experts' knowledge consists of a mixture of textbook knowledge, security obligations and laws, as well as experiences comprising typical and exceptional cases. Especially for novices, it is difficult to find the relevant information to cope with a particular problem.

In this section, we clarify requirements on security knowledge. Based on this, we review several security taxonomies and ontologies to identify relevant security concepts being appropriate to document relevant security incidents.

### A. Requirements on Security Knowledge

Regarding security, we need to deal with the unknown unknowns [26]. This refers to the problem that we cannot say which knowledge will be relevant in the future. Moreover, security knowledge is not necessarily limited and can change rapidly. Even established best practices can become out-dated. For example, a delay after entering a wrong password several times in order to login might be sufficient for local computers, but could facilitate a denial-of-service (DoS) attack on a web-based infrastructure. To cope with these problems, security knowledge needs to be maintained. This means that knowledge elements must be added or modified by humans iteratively. Therefore, it has to be represented in a symbolic manner instead of being modeled by statistical means (e.g. Naive Bayes). To enable semiautomatic security assessment, it needs to be suitable for computer-based processing.

### B. Security Ontology and Concepts

In our approach, security knowledge consists of security incidents enriched with additional information retrieved from natural language documents such as security guidelines and obligations. An incident is an attempt to violate security policies or to gain unauthorized access to data [15].

In recent years, various taxonomies and ontologies for modeling incident-centric security knowledge have been proposed. Since security knowledge is widely used, we focused on publications from the area of threat modeling, risk analysis, computer and network security, software vulnerabilities and information security management. Moreover, we consider

TABLE I
PRIMARY STUDIES AND THEIR PRINCIPAL SECURITY CONCEPTS

| Publication | Principal Security Concepts |
|---|---|
| Howard et al. [18] | Action, Target, Access, Tool, Vulnerability, Result, Objective, Attacker |
| Jung et al. [21] | Asset, Vulnerability, Threat, Security Control, Risk Probability, Asset Value, Impact, EC environment |
| Mouratidis et al. [29] | Constraints, Secure Entity (goals, tasks, resources), Secure Dependency |
| Undercoffer et al. [42] | Attack, System Component, Input, Consequence, Means, Location |
| Alvarez et al. [3] | Entry Point, Vulnerability, Service, Action, Input Length, HTTP Headers, HTTP Verb, Target, Scope, Priviliges |
| Swiderski et al. [40] | Asset, Entry Point, Trust Level, Attack, Attacker, Vulnerability, Countermeasure |
| Herzog et al. [16] | Asset, Threat, Vulnerability, Countermeasure |
| Tsoumas et al. [41] | Asset, Risk, Threat, Attack, Threat Agent, Vulnerability, Impact, Countermeasure, Controls, Security Policy, Stakeholder |
| Karyda et al. [22] | Asset, Countermeasure, Objective, Person, Threat |
| Barnum et al. [5] | Vulnerability, Weakness, Method of Attack, Attack Consequence, Attacker Skill, Solution and Mitigation, Resource, Context |
| Fenz et al. [12] | Asset, Organization, Security Attribute, Threat, Threat Source, Threat Origin, Vulnerability, Control, Severity Scale |
| Elahi et al. [10] | Vulnerability, Effect, Attack, Security Impact, Malicious Goal, Attacker, Countermeasure, Malicious Action, Component, Actor |
| Simmons et al. [38] | Attack Vector, Operational Impact, Defense, Informational Impact, Target (Network, Application, etc.) |
| Guo et al. [13] | Attack, Countermeasure, Consequence, Attacker, Vulnerability, IT Product |
| Miede et al. [28] | Attack, Countermeasure, Asset, Vulnerability, Threat, Security Goal |
| Eichler [8] | Asset, Threat, Damage Scenario, Protection Requirements, Safeguard, Module |

publications covering information systems, cyber-physical systems, distributed systems and agent-based systems. We ignored earlier taxonomies because they have been considered in recent publications. To cope with security issues in requirements engineering, we rejected publications which mainly focus on technical security aspects of systems (e.g. protocols, encryption algorithms). Identified primary studies and their principal security concepts are presented in Table I. Based on these concepts, we extracted a minimal knowledge structure, which can be found in each of the regarded taxonomies in one or the other way. In Fig. 2, the regarded security concepts and their relationships are presented. Based on the reviewed publications, we define the following concepts to model incidents and illustrate them with examples from the iTrust project:

- An *asset* is an item of interest worth being protected (e.g. password, medical identification number, patient and office visit information).
- *Entry points* define the interfaces to interact with the system. They provide access to assets (e.g. login website, health record, email, input field).
- A *trust level* describes which role has accesss to an asset using a specific entry point (e.g. patient, health care personnel, administrator).
- *System components* model the regarded system focusing on assets and entry points. This includes hardware as well as software components (e.g. database, logging).

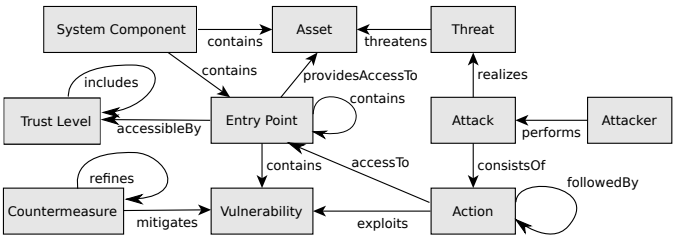


Fig. 2. Overall structure of the security knowledge. High level concepts and their relationships are presented. The relation *followedBy* marked with an asterix represents an ordered list in a simplified form.

- An *attack* is a sequence of malicious *actions* that are performed by an attacker aiming at assets (e.g. cross-site scripting, denial-of-service attack).
- A *vulnerability* is a system property that facilitates unintended access or modification of assets. It violates an explicit and implicit security policy. Entry points may have or provide access to vulnerabilities (e.g. improper neutralization of input, missing encryption of sensitive data).
- A *threat* is the possibility to perform a successful attack on a specific asset. Successful attacks exploit at least one vulnerability to cause damage (e.g. execute unauthorized code or commands, expose sensitive data).
- A *countermeasure* mitigates a certain threat by fixing the respective vulnerability (e.g. input validation, encryption of sensitive data).

Based on our aforementioned requirements, we decided to use ontologies to organize security knowledge. Moreover, reasoning and ontology query languages allow to have unified access to knowledge. Because security knowledge can be invalidated over time, we need to recognize when contradictory data is included. For this purpose, reasoning techniques help us to find such knowledge.

### C. Knowledge Acquisition and Incident Documentation

To support vulnerability prevention for requirements, incidents must be documented and enriched with security knowledge which is used to derive effective mitigation strategies. This knowledge can be retrieved from various sources such as security guidelines and standards, best practices as well as vulnerability catalogs. In our approach, an incident is modeled using corresponding actions which refer to several entry points, assets, and trust levels (see Fig. 2). Moreover, the vulnerability which has been exploited by an action has to be documented. For this purpose, mature incident documentation approaches exist [2]. This is not within the scope of this paper. Instead, we focus on refinement of the gathered incidents in order to improve security assessment of requirements.

## IV. Security Assessment in Requirements Engineering based on Heuristics

A security assessment is a systematic technique to analyze the current security status of the system under development [43]. This also includes identifying vulnerabilities in requirements. They are usually written in natural language to facilitate communication among different stakeholders. Assessing these documents with respect to security is a human-intensive and time-consuming task. Moreover, occurred security incidents may reveal new vulnerabilities which have not been considered by the requirements engineer yet. As a consequence, all requirements have to be assessed again taking the discovered weakness into account.

As an example, QR codes, which are used along with mobile devices (e.g. smart phones, tablets), introduced a new attack vector. QR codes are machine-readable barcodes which encode information about an item. To a large extent, they are used on advertising posters and contain links to the vendor's website. As previously reported [19], attackers stick manipulated codes over regular ones linking to malicious websites. People who used these codes without checking the target link infected their mobile devices. Based on this incident report, a requirements engineer needs to identify all use cases of the regarded system where QR codes are used. These findings are then analyzed in order to plan effective mitigation strategies.

This example also illustrates that newly discovered attack techniques and technological progress constitute new challenges. Regarding long-living software systems, the problem becomes even more challenging. In order to decrease the delay between discovering new incidents and assessing the specification, semiautomatic assessment techniques in requirements engineering are desired.

Security assessment of requirements comprises information about the security problem and references to effective mitigation strategies. For this purpose, we developed a heuristic to identify reported vulnerabilities and their variations in natural language requirements based on revealed security incidents.

### A. Towards Heuristics in Requirements Engineering

In this paper, a heuristic is defined as an analytical method to assess requirements with respect to security. Heuristics are based on hypotheses and have to cope with incomplete or unknown knowledge. Heuristic findings are suboptimal which means that in many cases they are correct, but there are false positives. The challenge is to develop heuristics that identify all intended situations reliably and, thus, reduce the number of false negatives. This gets even more challenging for long-living systems due to evolving assumptions and hypotheses. As a consequence, an initial heuristic may perform well but gets worse later on.

In our previous work, we tested different mechanisms to establish helpful heuristics for requirements engineering. In our first approach, we attempted informed guessing from experiences or from literature (e.g. no passive voice, not too long sentences) [24]. Based on this, we codified security experiences by statistical means [37]. For this purpose, we trained a Bayesian classifier with pre-evaluated security material. The classifier performed well for identifying security-relevant requirements automatically [17], [37]. It bridges the gap between security best practices and the lack of security experience among requirements engineers. However, detailed information about the security issue has not been provided by the approach. In this paper, we extend our previous work by using security knowledge to determine detailed security information from use cases. Additionally, we focus on the fact that in long-living systems security knowledge changes.

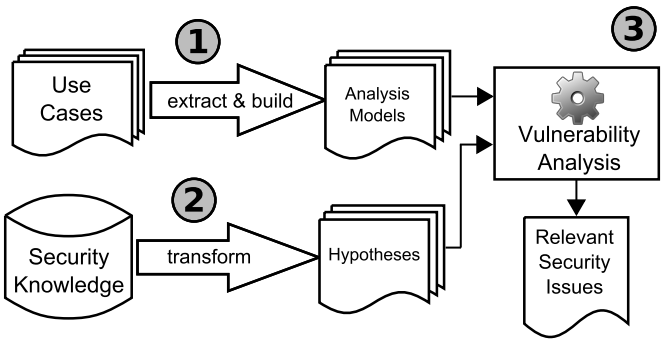


Fig. 3. Security assessment based on modeled security knowledge.

### B. Overview of the Proposed Security Assessment Approach

The aim of our approach is not to replace established and well-suited security assessment methods, but to support them by providing intermediate feedback to the requirements engineer. Therefore, our approach reveals potential security threats and vulnerabilities for given use cases. It takes the scenario description of the use cases as well as security knowledge as input. A brief overview of our assessment approach as illustrated in Fig. 3 is presented in the remaining section. Further details are explained in Sec. V.

In step 1, each use case written in natural language is transformed to a separate analysis model named *security abstraction model*. A security abstraction model consists of an ordered list of steps and represents the usage scenario with respect to security. Each step is described by a set of the following attributes: (1) system components, (2) entry points, (3) assets and (4) trust levels. To infer these attributes from use cases, natural language processing [4] is used. In step 2, hypotheses for the heuristic are built up from security knowledge. Therefore, each documented security incident is also transformed to a separate security abstraction model. The hypotheses are used in step 3 to assess the analysis model extracted from the given use cases. The idea is to detect suspicious sequences within the use cases which have been seen in incidents before. For example, an incident is reported that an attacker obtained sensitive data by intercepting unencrypted emails. Based on this report, a use case is marked as vulnerable if sensitive data (e.g. passwords) are sent via email. On this account, semantic similarity between the steps of the corresponding security abstraction models is derived.

Our approach returns a set of hypotheses which fits best with the given use cases. Hypotheses are used to infer the underlying threat as well as possible countermeasures from the security knowledge. Moreover, additional security information (e.g. security guidelines and standards, vulnerability databases, etc.) is attached to threats and countermeasures which can be used to analyze the vulnerability in detail.

## V. Details of our Security Assessment Approach

To compare information from different knowledge sources, it needs to be represented in a uniform manner. For this purpose, we use security abstraction models as a unified representation to encode use cases and security incidents. The proposed security analysis is defined on the basis of the security abstraction models.

Since we consider use cases written in natural language, our approach relies on natural language processing. To cope with ambiguities and imprecision of natural language, we introduce a linguistic approach to determine the semantic similarity between words. Semantic similarity quantifies how much a word *A* has a similar meaning to another word *B* although they are syntactically different [32]. Regarding the health care domain, the term *patient*, which refers to a trust level in iTrust, is semantically similar to the term *sick person* or *hypertensive*. Since we only consider concepts as presented in Sec. III, it is sufficient to focus on nouns. Thus, this enables identifying possible variations of vulnerabilities in requirements and increases the efficiency of our approach.

### A. Semantic Similarity Computation for Nouns

The goal is to measure the semantic similarity between nouns based on the structure and content of WordNet. In WordNet nouns are organized in hierarchies [11]. For this purpose, we adapt the approach presented in [20] which is based on the information content of the lowest common subsumer (LCS). The LCS is the concept in a tree-like lexical taxonomy, which has the shortest paths from the two concepts compared. According to WordNet, the term *sick person* is the LCS of the medical terms *diabetic* and *epileptic*. If the information content of the LCS is above a pre-defined threshold, it is too general to represent both concepts. Otherwise, both concepts are semantically similar.

To derive the LCS for the given concepts, the paths are derived by using their hypernyms listed in WordNet. A hypernym is a generic term used to describe a whole class of specific terms [44]. For example, the term *sick person* is the hypernym of the term *patient*. The information content of the LCS is defined as the sum of all concepts which have the LCS in common. For large word taxonomies such as WordNet, this is very CPU-intensive to derive. To overcome this problem, the information content of the LCS is approximated. On this account, for each concept on the paths the number of all direct hyponyms is summed up including all direct hyponyms of the LCS. A hyponym is the specific term used to describe a member of a class [44]. For example, one hyponym of *patient* is the term *hypertensive*.

### B. Building the Security Abstraction Models

To assess security as presented above, the usage scenario of use cases as well as security incidents must be transformed to security abstraction models.

*1) Determining Abstractions from Use Cases:* Use cases depict high-level functionality of the system. From a security perspective, a use case specifies which assets are accessed by which actors using certain entry points.

To derive security abstraction models, each use case undergoes a series of steps. Firstly, the description of each scenario step is enriched with syntactic information. Syntactic information specify whether a word is a noun, verb, or adjective. Therefore, a statistical part-of-speech tagger as presented in [35] is applied. Secondly, nouns and compound nouns are extracted and stemmed for each scenario step. Thirdly, extracted nouns are assigned to the attributes (e.g. system component, asset, entry point, trust level) of the respective security abstraction model. For example, the third use case of the iTrust system describes how user authentication is applied. Here, the actor of the use case is a *user* (trust level) who must enter her *medical identification number* and *password* (assets) into the *login form* (entry point) of the application. In iTrust, a user is a patient, a health care personnel, a public health agent, and so on. To assign extracted words to the attributes, the modeled security knowledge serves as a glossary.

Finally, abstracted scenario steps of a use case are arranged in an ordered sequence whereby the precondition is the first element followed by the trigger. The postcondition is put at the end of the sequence. Alternative paths in use cases are considered as separate security abstraction models. Moreover, loops and conditional clauses contained in usage scenarios are currently not evaluated.

*2) Extracting Hypotheses from Security Knowledge:* To derive hypotheses, the overall structure of the proposed security knowledge as presented in Sec. III is used. A hypothesis encodes incidents according to the security abstraction model. For this purpose, threats assigned to an attack (see relation *realizes*) are used to obtain assets which are threatened (see relation *threatens*). Actions of an attack are arranged chronologically using the relation *followedBy*. Each action represents a step in the security abstraction model. Actions refer to entry points which are used to access assets. System components, assets, and trust levels assigned to an entry point are used as attributes respectively. For example, an incident is reported that an attacker obtained passwords by trying words from a dictionary. Therefore, she enters each word into the login form of the application. In the corresponding security abstraction model the asset is set to *password*, the entry point is set to *login form*, and the trust level is set to *unknown* because no further information about the attacker is provided.

### C. Measuring Similarity between Security Abstraction Models

To identify vulnerabilities in use cases, the corresponding security abstraction models are evaluated for the occurrence of the steps encoded in the hypotheses. A similar sequence of steps may indicate that the system described by the use case

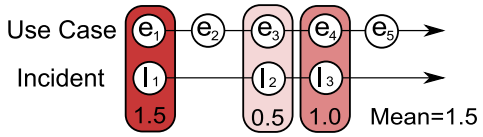


Fig. 4. Alignment between the security abstraction models derived from a use case and a security incident.

is vulnerable with respect to the respective security incident. Concerning this, steps do not have to follow immediately. This means that any two adjacent steps within a hypothesis are separated by an arbitrary number of steps within the security abstraction model of the use case. The maximum match of two sequences is defined as the largest number of similar steps which occur in the same order (sequence alignment) as illustrated in Fig. 4. The numbers in this figure represent the degree of semantic similarity between steps. The corresponding mean value is used to evaluate whether a relevant match has been found.

To derive the alignment in our approach, we utilize the Needleman-Wunsch algorithm [30]. The smallest unit of comparison is a pair of steps, one from each security abstraction model. All possible pair combinations are determined and stored in the corresponding cell of a two-dimensional matrix (dynamic programming). For this purpose, the semantic similarity computation as described in Sec. V-A is used. Every possible comparison can be represented by pathways through the matrix. The maximum match is represented by the pathway for which the mean value of assigned cell values is largest. If the mean value is above a given threshold, a possible vulnerability have been found for the given use case.

Regarding the security abstraction models from our examples in Sec. V-B1 and Sec. V-B2, the asset *password* and the entry point *login form* which have been extracted from the usage scenario are equal to the corresponding attributes of the incident. The trust level *unknown* from the incident contains the trust level *user* from the use case. As a result, the hypothesis matches with the security abstraction model of the use case which is therefore marked as vulnerable.

## VI. Extracting Security Knowledge

In order to develop practical knowledge bases, it is necessary to acquire, understand, encode, and organize human knowledge appropriately. This is important for requirements engineering because it is usually not possible to predefine a definitive body of security knowledge for a certain domain. One reason is that some domains are partially understood and need to be clarified over time. Thus, learning security knowledge helps to improve security of long-living software systems. Learning in our context means (1) acquiring new knowledge or (2) modifying, reinforcing, and refining existing knowledge. The challenge is that we only have a few incidents as problem instances to learn from. For this reason, the aim of this method is to iteratively learn security knowledge.

We define knowledge as the expertise stored in a person's mind which has been obtained by interacting with the person's environment [36]. To acquire security-relevant knowledge efficiently, we need an elicitation method which is seamlessly integrated into the security analysis workflow. On that account, we propose a method that is built upon the heuristic findings which have been achieved by using our security assessment approach. These findings highlight possible vulnerabilities within the use cases and consist of truly and falsely classified use cases (true and false positives). True positives mainly contain additional knowledge about system components, assets, entry points, and trust levels. However, false positives are used to specify terms for the regarded domain. This means that all meanings of a term, which are not valid in a given domain, are excluded explicitly. To extract knowledge, the requirements engineer has to evaluate all vulnerable use cases. The effort to refine knowledge decreases over time because knowledge of a given system improves.

To extract additional knowledge from true positives, natural language parsing and syntax tree analysis is used. The extraction process is guided by applying the proposed knowledge structure as presented in Sec. III. Input of our method is a set of heuristic findings. As presented in Sec. V-C, they consist of suspicious scenario steps of the use cases which may compromise the security of the system. Each finding undergoes a series of steps. Firstly, the linguistic structure (syntax tree) of the regarded scenario step is determined. Secondly, the syntax tree is analyzed with respect to the attributes (system component, asset, entry point, and trust level) of the corresponding incident. For example, if in the description "The user enters an identification number and a pin." the asset *pin* is identified, it can be implied that *identification number* is also an asset because it has a linguistic dependency to *pin*. Thirdly, the requirements engineer is questioned whether the new knowledge should be added to the knowledge base. Moreover, added knowledge can be enriched by a security expert with additional security information (e.g. security standards and guidelines).

In order to specify terms for a certain domain, false positives are mainly considered. For this purpose, falsely classified scenario steps and their semantic relationships to the respective incident are used. Attributes (system component, asset, entry point, and trust level) of the incident and their corresponding match within the scenario step are evaluated. If the semantic similarity is below a pre-defined threshold the requirements engineer is questioned whether both attributes mean the same in the given domain. Thus, the requirements engineer can actively choose which terms must be excluded from semantic similarity computation as presented in Sec. V-A.

Security knowledge extraction performed by humans as well as refinement of knowledge is also prone to ambiguities. However, we consider the identification and dissemination of vulnerabilities a higher value.

## VII. iTrust Case Study

The aim of this case study is to evaluate whether our approach can support requirements engineers. In our previous work [17], [37], we used a Naive Bayes classifier to derive

whether a requirement is security-relevant or not. Thus, we evaluate the performance of our approach in contrast to Naive Bayes classifiers. Moreover, we also consider Support Vector Machines (SVM) and k-Nearest Neighbors (k-NN) since they perform well in many machine learning tasks.

### A. Case Study Design

Ten initial use cases have been selected to set up the security knowledge according to Sec. III. The selection criterion is that at least each use case has a different actor and thus covers a different functionality of the iTrust system. Since iTrust does not have documented security incidents, we prepared misuse cases (MUC) as described in [1]. Therefore, we focused on the technological domain because we are no health care experts.

In order to evaluate whether the performance of our heuristic is improved through knowledge refinement, we performed two iterations. For the first iteration, we selected 35 iTrust use cases. Together with the initial use cases, they have been assessed using our approach. Afterwards, we compared the heuristic findings with our manual assessment which we have done before. Considering true positives and false positives, we refined the security knowledge (see Sec. VI). Based on the refined knowledge, we assessed all use cases in the second iteration.

*1) Misuse Cases:* From analyzing use case 1 (UC1), we obtain misuse case 1 (MUC1). In UC1, an email, which contains the initial password to log into a user account of the iTrust system, is sent to the user which has been created by the health care personnel. In the MUC1, the email is intercepted by an attacker which thus obtains sensitive user information. The attacker uses this information to gain access to the user account. Therefore, hijacking user accounts becomes possible. To mitigate this vulnerability, emails containing sensitive information must be encrypted.

Analyzing UC6 leads to MUC2. In MUC2, the patient view with respect to the address field contains a vulnerability so that cross-site scripting (XSS) attacks become possible. This is one of the most dangerous vulnerabilities in web applications. XSS becomes possible through improper neutralization of input. Attacker can inject malicious browser-executable content into the patient view to steal sensitive information (e.g. medical identification number or password). To mitigate this vulnerability, all use cases need to be identified which render demographic information. Identified use cases and their corresponding code can be checked whether they sanitize input properly.

*2) Security Knowledge:* Based on the initial use cases, an appropriate security ontology is set up as proposed in Sec. III. For this purpose, terms from the iTrust glossary (health care domain) are used as well. Furthermore, system components, assets, trust levels and entry points of the misuse cases are added as listed in Table II.

*3) Naive Bayes, SVM, and k-NN:* To set up Naive Bayes, SVM, and k-NN in the training phase, the initial use cases are labeled with respect to the misuse cases. If a misuse case is related to a use case in any way, the use case is labeled *sec*. Otherwise, it is labeled *non-sec*. Here, words of the given use case descriptions are not rated separately since it is usually not easy to decide which words indicate non-security information.

TABLE II
EXTRACT OF SECURITY KNOWLEDGE USED IN ITRUST CASE STUDY.

| Concept | Individuals |
|---|---|
| **MUC 1** | |
| Asset | initial password, security key |
| Entry Point | email |
| Trust Level | user |
| **MUC 2** | |
| Asset | address |
| Entry Point | address field, health record, view, display |
| Trust Level | patient, health care personnel (HCP) |

### B. Results

The quality of our approach is measured in terms of accuracy (ACC), false positive rate (FPR) and false negative rate (FNR). The accuracy is defined as the degree of correctly assessed use cases with respect to all use cases. The FPR measures the rate of use cases which have been falsely assessed as security-relevant. In contrast, the FNR measures the rate of use cases which have been falsely assessed as not security-relevant. A low FNR is desired since it means that at least all security-relevant use cases have been found correctly.

*1) First Iteration:* In Table III, results of the first iteration with 44 uses cases are presented. Since one incident was not enough to achieve reasonable results, we trained one classifier for both misuse case. As a result, Naive Bayes, SVM, and k-NN have a high FRN which means that a lot of vulnerable use cases were overlooked. Regarding SVM, the FNR is 1.0 which means that no vulnerable use cases were identified. For that reason, we cannot refine knowledge based on heuristic findings. Thus, SVM is not considered in the second iteration.

Our approach identified all five use cases which are related to MUC1 in any way. Thus, the recall of our approach is 1.0. Regarding MUC1, the FNR is 0.0 which mean that no use case was overlooked. Moreover, our approach identified 11 out of 13 use cases which are related to MUC2 in any way. This implies that the recall is around 0.85. However, FNR is 0.15

TABLE III
EVALUATION RESULTS (ACC = ACCURACY, FPR = FALSE POSITIVE RATE, FNR = FALSE NEGATIVE RATE)

| | | ACC | FPR | FNR |
|---|---|---|---|---|
| **1st Iteration (n=44)** | | | | |
| Our Approach | MUC 1 | 0.90 | 0.10 | 0.00 |
| | MUC 2 | 0.64 | 0.55 | 0.15 |
| Naive Bayes | MUC 1/2 | 0.61 | 0.00 | 0.89 |
| SVM | MUC 1/2 | 0.57 | 0.00 | 1.00 |
| k-NN | MUC 1/2 | 0.57 | 0.15 | 0.83 |
| **2nd Iteration (n=55)** | | | | |
| Our Approach | MUC 1 | 0.98 | 0.00 | 0.14 |
| | MUC 2 | 0.84 | 0.14 | 0.23 |
| Naive Bayes | MUC 1/2 | 0.71 | 0.11 | 0.67 |
| k-NN | MUC 1/2 | 0.76 | 0.00 | 0.68 |

which means that two vulnerable use cases were not identified. The number of false positives is 17 which means that more false positives were found than true positives. Nevertheless, 14 out of 44 use cases can be ignored (true negatives) which reduce the effort for manual assessment by a third. The reason why our approach performed worse in MUC2 is that the asset *address* as well as the entry point *address field* are not very specific for the health care domain and thus are similar to many words semantically. Thus, it is necessary to define the meaning of these terms more precisely.

*2) Knowledge Refinement:* Based on Sec. VI, we extracted additional knowledge from the heuristic findings and thus refined the applied security knowledge. On that account, we considered all use cases which were identified as vulnerable. True positives were used to add additional knowledge. To specify used terms, false positives were mainly considered.

Regarding UC3, the term *security answer* was added as a valuable asset which also should not be sent via email. Moreover, UC25, UC30 and UC40 were used to specify the meaning of the asset *password* with respect to the iTrust system. It was added that *password* is not a *word*, a *number*, or a *diagnosis*.

Analyzing UC32, the entry point *patient report* was added since it contains the patient's address and views it on a webpage. As mentioned above, the asset *address* is not specific for the health care domain. For this purpose, UC3, UC8, UC10, UC32 are used to define that *address* is not an *IP address*, an *email address*, an *access*, a *link*, a *prescription*, or a specific health care *code*.

To re-train the Naive Bayes as well as k-NN classifier, we added the true positives to the training set and labeled them as *sec*. For Naive Bayes two use cases and for k-NN three use cases were considered. Furthermore, we also added the false positive and labeled them as *non-sec*. Thus, four falsely classified use cases were added for k-NN. Naive Bayes produced no false positives.

*3) Second Iteration:* The results of the second iteration with 55 use cases are presented in the lower part of Table III. Regarding Naive Bayes and k-NN, the FNR are lower compared to the first iteration. But only two additional use cases were identified as vulnerable by Naive Bayes. For k-NN only one additional use case was found. As a result, Bayes, SVM, and k-NN are not applicable to identify the majority of vulnerabilities based on incidents in natural language requirements.

Regarding our approach, however, the accuracy is good for both misuse cases. Moreover, the FPR for MUC2 decreased from 0.55 to 0.14. Indeed, no false positives were produced for MUC2. Thus, knowledge refinement has a significant impact on the heuristic findings regarding FPR. The FNR, however, has increased slightly compared to the first iteration. But only true positives, which have been already considered by the requirements engineer in the first iteration, are affected.

### C. Discussion

Based on the iTrust case study, we discuss the achieved results with our research questions. To model security incidents which are used in our security assessment approach, a knowledge structure has been presented in Sec. III. Based on this structure, the security abstraction model has been defined. According to **RQ1**, the results of the iTrust case study indicate that the proposed concepts are sufficient to represent use cases and incident in a uniform manner. Moreover, the proposed knowledge structure is small enough to document incident without requiring too much human effort. Only system components, assets, entry points, and trust levels for each step of the incident must be documented.

Answering **RQ2**, identification of vulnerable use cases has been evaluated using the iTrust case study. Since reported incidents are not available for iTrust, we derived two misuse cases from the given requirements. Moreover, we also compared our approach to methods which we developed in our previous research. As a result, Naive Bayes, SVM, and k-NN performed very weak. One reason is that training data is sparse because only a few misuse cases are available to learn from. However, this is common in our usage scenario. In comparison, our security assessment approach achieved better results in terms of accuracy, false positive rate and true positive rate. For both misuse cases the false negative rate of our approach is acceptable. Only two out of 13 use cases for MUC2 were overlooked. Thus, we can state that our approach is sufficient to identify vulnerable use cases reliably. Although the results are good for the case study, further studies are needed in order to generalize results.

As the results of the second iteration compared to the first iteration imply, knowledge refinement can reduce the number of false positives sufficiently. In order to answer **RQ3**, we presented an extraction method guiding the requirements engineer to refine terms and properties and add new ones to the security knowledge. Therefore, it leverages heuristic findings obtained by our assessment approach. Although it requires some effort to handle case by case at the beginning, the knowledge saturates after a few iterations and the effort to refine knowledge decreases. Regarding long-living software system, we consider this a valuable investment. According to the results of the iTrust case study, our method is suitable to refine security knowledge which is organized according to **RQ1**. This supports reusing security knowledge gained during the development of security-critical software and feeding it back into requirements engineering.

Although the results indicate that our approach works, we have identified some limitations while conducting the iTrust case study. Regarding the security abstraction model, it is not feasible to encode attributes which are described in the use case implicitly. As an example, the description "The user enters the password." contains the trust level *user* and the asset *password*, but the entry point is implicitly encoded in the verb *enter*. If we know that the regarded system is a web application, we are able to imply that the entry point may be a *website*. Computer-based reasoning approaches, however, need this knowledge modeled beforehand. Moreover, the proposed knowledge structure cannot model simultaneous actions. For example: "While entering the required password, only the last

character is shown in clear text." In some cases an incident may affect more than one use case. To identify these incidents, the interrelationship between use cases must be considered.

Nevertheless, our assessment approach gains appropriate results with reasonable computation costs. They can be considered as an initial indication of the presence of security loopholes followed by an in-depth security analysis.

## VIII. Related Work

AlHogail and Berri [2] propose the development of an architecture sustaining security knowledge within an organization. The architecture aims to manage tailored security processes, policies, and solutions. The goal is to capture and share security knowledge in order to efficiently react on security incidents and decrease dependencies on security experts. To capture security incidents as well as countermeasures, a predefined report template is used. Reports are analyzed manually to establish rules that are stored in an organization-wide knowledge base. Tsoumas and Gritzalis [41] suggest a security management approach for information systems which builds upon security knowledge gathered from various information sources. Raskin et al. [33] present an ontology-driven security approach. Ontologies are used to organize security knowledge (e.g. attacks and countermeasures, etc.) retrieved from natural language sources. Other approaches dealing with similar ontologies to manage security knowledge properly are presented in [23], [6]. In these approaches the ontology is mainly set up beforehand considering widely-accepted standards and information about the infrastructure of the information system as well as established policy documents.

Nuseibeh et al. [31] described their experiences applying a security requirements analysis. For this purpose, they developed a framework and associated tools that help to analyze the context systematically in which the system operates. It therefore offers different forms of structured argumentation. In Haley et al. [14], an approach to support security requirements elicitation and analysis is proposed. The method is based on constructing a system context using a problem-oriented notation. The system context is validated against the security requirements that are modeled as formal constraints. The approach reveals whether security requirements cannot be satisfied in the context or that the context is not sufficiently defined. Although the approach is very powerful, it requires a certain level of expertise to construct the context and understand the analysis results. To identify security requirements for certification and accreditation, Lee et al. [25] developed an approach based on extraction of relevant concepts form regulatory documents which are used to build up a problem domain ontology. Elahi et al. [9] proposed a vulnerability-centric requirements engineering framework for assessing the risk of vulnerabilities exploitation. Thus, this approach enables security requirements elicitation and analysis based on vulnerabilities. For this purpose, an extension of the i* metamodel to incorporate security-relevant concepts (e.g. attacks, countermeasures, vulnerabilities) is proposed. Moreover, a modeling process is proposed to attain the analysis model. Sindre and Opdahl [39] have been presented an approach to identify possible threats and analyze risks by eliciting misuse cases based on pre-defined templates. Another well-structured analysis method is to conduct security assessment by means of attack trees. An attack tree represents known attacks and their countermeasures in a tree structure. In Byres et al. [7], attack trees are used for assessing vulnerabilities in SCADA systems successfully. Jung et al. [21] proposed a case-based reasoning approach to incorporate past security accidents in risk analysis of e-commerce system. To apply this approach, the problem must be formulated in the well-structured case representation which requires additional effort.

In most approaches, evolution of security knowledge is not considered sufficiently. We are not aware of development approaches dealing with security knowledge and its evolution in a systematic way. To support requirements engineers by incorporating security knowledge semiautomatically, has not been considered in most approaches. Moreover, evolution of the regarded software system as well as knowledge changes has been insufficiently considered yet.

## IX. Conclusions and Future Research

Maintaining information systems with many natural language requirements is a laborious task. Security related parts of a software have to be changed when requirements change or when assumptions about the system context do not hold any longer due to occurred security incidents.

In this paper, we presented a security assessment approach for supporting maintenance of long-living information systems independent of the process model, domain, or technology in use. According to our research questions, our approach is suitable to identify vulnerabilities in natural language requirements based on reported security incidents. It therefore provides support for avoiding repetition of security flaws which already occurred before. Moreover, we presented a method to refine or adapt security knowledge and to make it more suitable for future security assessments. In the iTrust case study, we have shown that our approach performs better than other approaches (Naive Bayes, SVM, k-NN). Although we only focus on technical aspects of security within this paper, our approach is capable of incorporating human-related aspects as well.

The success of our assessment approach highly depends on the quality of the security knowledge. Whereas detailed knowledge leads to more helpful information, it is expensive to retrieve and to model. To use our approach in an industrial setting, we have to evaluate a manageable level of detail that is used to document incidents. Moreover, we need to investigate whether intermediate feedback about security issues in requirements improves security of the software system. For this reason, we plan to conduct an industrial case study.

The presented approach is a foundation for the other parts of SecVolution. Identifying security issues in requirements and refining security knowledge in the requirements engineering process is important for our security aware evolution approach of long-living software systems.

## ACKNOWLEDGMENT

This research is funded by the DFG project SecVolution (JU 2734/2-1 and SCHN 1072/4-1) which is part of the priory program SPP 1593 “Design For Future - Managed Software Evolution”.

## REFERENCES

[1] I. Alexander. Misuse cases: Use cases with hostile intent. *IEEE Software*, 20(1):58–66, 2003.

[2] A. AlHogail and J. Berri. Enhancing it security in organizations through knowledge management. In *Proc. of the Int. Conference on Information Technology and e-Services (ICITeS)*, pages 1–6. IEEE, 2012.

[3] G. Álvarez and S. Petrovic. A new taxonomy of web attacks suitable for efficient encoding. *Computers & Security*, 22(5):435–449, 2003.

[4] V. Ambriola and V. Gervasi. Processing natural language requirements. *Automated Software Engineering*, pages 36–45, 1997.

[5] S. Barnum and A. Sethi. Attack Patterns as a Knowledge Resource for Building Secure Software. In *OMG Software Assurance Workshop: Cigital*, 2007.

[6] P. Belsis, S. Kokolakis, and E. Kiountouzis. Information systems security from a knowledge management perspective. *Information Management & Computer Security*, 13(3):189–202, 2005.

[7] E. Byres, M. Franz, and D. Miller. The use of attack trees in assessing vulnerabilities in SCADA systems. In *Proc. of the Int. Infrastructure Survivability Workshop*, 2004.

[8] J. Eichler. Lightweight modeling and analysis of security concepts. In *Proc. of the Third Int. Conference on Engineering Secure Software and Systems*, ESSoS, pages 128–141. Springer, 2011.

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

[10] G. Elahi, E. Yu, and N. Zannone. A modeling ontology for integrating vulnerabilities into security requirements conceptual foundations. In *Proc. of the 28th Int. Conference on Conceptual Modeling*, pages 99–114, 2009.

[11] C. Fellbaum, editor. *WordNet: an electronic lexical database*. MIT Press, 1998.

[12] S. Fenz and A. Ekelhart. Formalizing information security knowledge. In *Proc. of the 4th Int. Symposium on Information, Computer, and Communications Security (ASIACCS)*, page 183. ACM, 2009.

[13] M. Guo and J. A. Wang. An ontology-based approach to model common vulnerabilities and exposures in information security. In *ASEE Southest Section Conference*, 2009.

[14] C. B. Haley, R. C. Laney, J. D. Moffett, and B. Nuseibeh. Security Requirements Engineering: A Framework for Representation and Analysis. *IEEE Trans. Software Eng.*, 34(1):133–153, 2008.

[15] S. Hansman and R. Hunt. A taxonomy of network and computer attacks. *Computers & Security*, 24(1):31–43, 2005.

[16] A. Herzog, N. Shahmehri, and C. Duma. An ontology of information security. *IJISP*, 1(4):1–23, 2007.

[17] S. H. Houmb, S. Islam, E. Knauss, J. Jürjens, and K. Schneider. Eliciting security requirements and tracing them to design: an integration of Common Criteria, heuristics, and UMLsec. *Requirements Engineering*, 15(1):63–93, 2009.

[18] J. D. Howard and T. A. Longstaff. A common language for computer security incidents. Technical report, Sandia National Laboratories, 1998.

[19] A. Jain and D. Shanbhag. Addressing security and privacy risks in mobile applications. *IT Professional*, (October):28–33, 2012.

[20] J. Jiang and D. Conrath. Semantic similarity based on corpus statistics and lexical taxonomy. In *Proc. of International Conference Research on Computational Linguistics (ROCLING)*, 1997.

[21] C. Jung, I. Han, and B. Suh. Risk analysis for electronic commerce using case-based reasoning. *Int. Journal of Intelligent Systems in Accounting, Finance & Management*, 8(1):61–73, Mar. 1999.

[22] M. Karyda, T. Balopoulos, L. Gymnopoulos, S. Kokolakis, C. Lambrinoudakis, S. Gritzalis, and S. Dritsas. An ontology for secure e-government applications. In *Proc. of the First Int. Conference on Availability, Reliability and Security*, pages 1033–1037. IEEE, 2006.

[23] S. Kesh and P. Ratnasingam. A knowledge architecture for IT security. *Communications of the ACM*, 50(7), 2007.

[24] E. Knauss, D. Lübke, and S. Meyer. Feedback-Driven Requirements Engineering: The Heuristic Requirements Assistant. In *Proc. of the 31st Int. Conference on Software Engineering (ICSE)*, pages 587 – 590, 2009.

[25] S.-W. Lee, R. Gandhi, D. Muthurajan, D. Yavagal, and G.-J. Ahn. Building problem domain ontology from security requirements in regulatory documents. In *Proc. of the Int. workshop on Software engineering for secure systems (SESS)*, pages 43–50. ACM, 2006.

[26] H. Mcmanus and P. D. Hastings. A Framework for Understanding Uncertainty and its Mitigation and Exploitation in Complex Systems. In *15th Annual In. Symposium of the Int. Council on Systems Engineering (INCOSE)*, pages 1–20, 2005.

[27] A. Meneely, B. Smith, and L. Williams. iTrust electronic health care system case study. In *Software and Systems Traceability*, pages 425–438. Springer, 2012.

[28] A. Miede, N. Nedyalkov, C. Gottron, A. König, N. Repp, and R. Steinmetz. A Generic Metamodel for IT Security Attack Modeling for Distributed Systems. *Int. Conference on Availability, Reliability and Security (ARES)*, pages 430–437, 2010.

[29] H. Mouratidis, P. Giorgini, and G. Manson. An ontology for modelling security: The tropos approach. In V. Palade, R. Howlett, and L. Jain, editors, *Knowledge-Based Intelligent Information and Engineering Systems*, volume 2773 of *LNCS*, pages 1387–1394. Springer, 2003.

[30] S. B. Needleman and C. D. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of molecular biology*, 48:443–453, 1970.

[31] B. Nuseibeh, C. B. Haley, and C. Foster. Securing the Skies: In Requirements We Trust. *IEEE Computer*, 42(9):64–72, 2009.

[32] T. Pedersen, S. Patwardhan, and J. Michelizzi. WordNet:: Similarity: measuring the relatedness of concepts. In *Demonstration Papers at HLT-NAACL*, pages 38–41. Association for Computational Linguistics, 2004.

[33] V. Raskin, C. F. Hempelmann, K. E. Triezenberg, and S. Nirenburg. Ontology in information security: a useful theoretical foundation and methodological tool. In *Proc. of the Workshop on New security paradigms*, NSPW, pages 53–59. ACM, 2001.

[34] T. Ruhroth, S. Gärtner, J. Bürger, J. Jürjens, and K. Schneider. Versioning and evolution requirements for model-based system development. In *Int. Workshop on Comparison and Versioning of Software Models (CVSM)*, 2014.

[35] H. Schmid. Probabilistic Part-of-Speech tagging using decision trees. In *Proc. of the Int. Conference on New Methods in Language Processing*, pages 44–49, 1994.

[36] K. Schneider. *Experience and Knowledge Management in Software Engineering*. Springer, 2009.

[37] K. Schneider, E. Knauss, S. Houmb, S. Islam, and J. Jürjens. Enhancing security requirements engineering by organizational learning. *Requirements Engineering*, 17(1):35–56, 2011.

[38] S. Shiva, C. Simmons, C. Ellis, D. Dasgupta, S. Roy, and Wu. AVOIDIT: A cyber attack taxonomy. Technical report, University of Memphis, August, 2009.

[39] G. Sindre and A. L. Opdahl. Eliciting security requirements with misuse cases. *Requirements Engineering*, 10(1):34–44, 2005.

[40] F. Swiderski and W. Snyder. *Threat Modeling*. Microsoft Press Corp., 2004.

[41] B. Tsoumas and D. Gritzalis. Towards an Ontology-based Security Management. In *Proc. of the 20th Int. Conference on Advanced Information Networking and Applications (AINA)*, volume 1, pages 985–992. IEEE, 2006.

[42] J. Undercoffer, A. Joshi, and J. Pinkston. Modeling computer attacks: An ontology for intrusion detection. In *6th Int. Symposium on Recent Advances in Intrusion Detection*, pages 113–135. Springer, 2003.

[43] K. H. Vellani. Vulnerability Assessments. In *Strategic Security Management: A Risk Assessment Guide for Decision Makers*, pages 85–107. 2007.

[44] WORDNET Webpage. Glossary of terms used in WordNet system , 2014.